k-Nearest Neighbor

- *Algorithm: k-Nearest Neighbor* (algo. 1)
- *Input:* training sample (with label), testing sample
- *Complexity:* $\mathcal{O}(n)$
- *Data structure compatibility:* vector
- *Common applications:* classification, regression

Problem. k-Nearest Neighbor

Given a labeled training set and a testing sample, use the k-nearest training sample to predict the the testing sample, under some distance definition and certain average definition.

Description

k-Nearest Neighbor, k-NN, is a widely-used supervised learning, meaning the training set is labeled explicitly before the training process. Its goal is to predict the testing sample, by the information from the k-nearest training samples of the testing sample. What needs to notice is that k-NN is a *lazy learning*, which means it never trains the model until the testing sample is consumed. It is when the algorithm gets the testing sample does the training process start.

The performance of k-NN will differ if different k and different distance is taken. Also, for the data with high dimensions of information, one prefers to do some dimension reduction before k-NN since k-NN cannot assure accuracy when the dimension is high.

k-NN can be used for classification and regression. For classification tasks, the testing sample will bear the label of the label in k-nearest neighbors with highest frequency. For regression, the testing sample will take the average of the k-nearest neighbors' labels.

Algorithm

k-Nearest Neighbor is a representative lazy learning, it only consumes time when a testing sample is coming. The pseudo-code of determining the label of testing sample is given in Algorithm 1.

For regression task, the average function $\mathit{average}$ on Line 16 of Algorithm 1 should be customized when tackling different problems.

Distance

The distance used for k-NN can vary. Generally speaking, the metric distance is used:

$$
\begin{aligned}
&\mathit{dist}(\mathbf{a}, \mathbf{b}) = \mathit{dist}(\mathbf{a}, \mathbf{b}) \\
&\mathit{dist}(\mathbf{a}, \mathbf{c}) \leq \mathit{dist}(\mathbf{a}, \mathbf{b}) + \mathit{dist}(\mathbf{b}, \mathbf{c}) \\
&\mathit{dist}(\mathbf{a}, \mathbf{b}) \geq 0 \\
&\mathit{dist}(\mathbf{a}, \mathbf{b}) = 0 \Leftrightarrow \mathbf{a} = \mathbf{b}
\end{aligned}
$$

Algorithm 1: k-Nearest Neighbor

Input : training samples $T = \mathbf{t_0}, ...\mathbf{t_{n-1}}$, the testing sample s, the argument k, the mode *mode*
Output: The predicted label *label*

```
set = { };
counter ← 0;
foreach t_i ∈ T do
    if dist(t_i, s) then
        set.append(t_i.label)
        counter ← counter + 1;
    end if
    if counter < k then
        break;
    end if
end foreach
if mode = classification then
    label ← The label appears the most times in set;
end if
if mode = regression then
    label ← average(label ∈ set)
end if
return label;
```

A sample of metric distance is the $p-$norm, also called Minkowski distance:

$$dist(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=0}^{d} (\mathbf{a}_i - \mathbf{b}_i)^p \right)^{1/p}$$

when $p = 2$, it is called Euclidean distance, and when $p = 1$, it is called Manhattan distance. Different distance has different performance on a certain problem, should be adjusted to the setting of problems.

Performance

The performance of k-Nearest Neighbor will significantly differ when different k and distance are taken. However, if the k samples are chosen adequately, and the distance is well-chosen, the performance of k-Nearest Neighbor will be good. An example of the nearest neighbor ($k = 1$) will be taken to illustrate the accuracy. [3]

Denote the label space as $\mathcal{Y}$, the testing sample as $\mathbf{x}$, the nearest neighbor as $\mathbf{z}$. Then the error can be represented as

$$P(err) = 1 - \sum_{c \in \mathcal{Y}} P(c|\mathbf{x}) P(c|\mathbf{z})$$

Suppose the samples are independently identically distributed, and there is always a training sample located in the δ-neighborhood of the testing sample while δ is arbitrarily small. Denote $c^* = \mathrm{argmax}_{c \in \mathcal{Y}} P(c|\mathbf{x})$ as the optimized classification, given by Bayesian optimize classifier. Then

$$\begin{aligned} P(err) &= 1 - \sum_{c \in \mathcal{Y}} P(c|\mathbf{x}) P(c|\mathbf{z}) \\ &\simeq 1 - \sum_{c \in \mathcal{Y}} P^2(c|\mathbf{x}) \\ &\leq 1 - P^2(c^*|\mathbf{x}) \\ &= (1 + P(c^*|\mathbf{x}))(1 - P(c^*|\mathbf{x})) \\ &\leq 2(1 - P(c^*|\mathbf{x})) \end{aligned}$$

The error of 1−Nearest Neighbor will not be greater than twice of the optimized error.

However, this performance is built on two assumptions: first is the adequate arguments k and the $dist()$ function is taken, second is the training samples are dense enough. The first requirement expects some good distance. The metric learning can help to find this adequate distance metric. [1] The second requirement will be break when the dimension is too high. When the dimension is increasing, the required training samples to satisfy the dense condition will increase exponentially [3]. Hence, one should perform dimension reduction, such as PCA or KPCA (one kind of non-linear dimension reduction; see [2]) to the training sample space before a k-NN is used.

References.

[1] Sam Roweis, Geoffrey Hinton, and Ruslan Salakhutdinov. "Neighbourhood component analysis". In: *Adv. Neural Inf. Process. Syst.(NIPS)* 17.513-520 (2004), p. 4 (cit. on p. 3).

[2] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem". In: *Neural computation* 10.5 (1998), pp. 1299–1319 (cit. on p. 3).

[3] Zhou Zhihua. *Machine Learning*. Tsinghua University Press, 2016 (cit. on pp. 2, 3).

Naive Bayesian Classification

- *Algorithm: Naive Bayesian Classification* (algo. 2)
- *Input:* training sample (with label), testing sample
- *Complexity:* $\mathcal{O}(n)$
- *Data structure compatibility:* vector
- *Common applications:* classification

Problem. Naive Bayesian Classification

In a classification task, if one wants to minimize the classification error, the posterior probability $P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x}}$ should be maximized. Since the whole space of $\mathbf{x}$ is too large to sample directly, an approximate estimate of $P(\mathbf{x}|c)$ is raised by naive Bayesian classifier, which supposes different attributes of $\mathbf{x}$ influence class-conditional probability independently.

Description

Naive Bayesian classification is a method to classify a testing sample, with some training sample with label is given. Thus, it is one kind of supervised learning. After the learning, it will give a prediction of the testing sample's label.

Naive Bayesian classification is based on the Bayesian optimal classifier, which implies a theoretically "best" classifier, i.e. with the lowest error, by $h^*(\mathbf{x}) = \arg\max_{c\in\mathcal{Y}} \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})}$. Naive Bayesian classifier supposes all attributes of $\mathbf{x}$ distributes to the class-conditional probability $P(\mathbf{x}|c)$ independently, then gives an estimate of the posterior and turns out to calculate out the $\arg\max$.

General Goal: Maximize Posterior Probability

In a classification task, if one wants to minimize the classification error, the posterior probability, of chosen c under the given condition $\mathbf{x}$, $P(c|\mathbf{x})$ should be maximized. Then, one will get the Bayesian optimal classifier [2]

$$h^*(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} P(c|\mathbf{x})$$

This $P(c|\mathbf{x})$ is called **posterior probability**, indicates the probability of label c appears when a sample $\mathbf{x}$ is already known. By Bayesian theorem, one can immediately get

$$h^*(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} P(c|\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})}$$

In this equation, observe that $P(\mathbf{x})$ is the same for all c. Hence, the optimization goal is to find an approximate estimate of $P(c)P(\mathbf{x}|c)$ and maximize it. Both the **prior probability** $P(c)$ and **class-conditional probability** $P(\mathbf{x}|c)$ should be estimated.

Finding Estimation

Estimate Prior

The prior $P(c)$ is easy to estimate, since the size of label space $|c|$ will not be extremely large and one can always sample enough training vectors (with label), to make the unbiased estimate

$$\widehat{P(c)} = \frac{|D_c|}{|D|}$$

(where D_c is the training samples of label c, and D is all the training samples) converge to $P(c)$ at any needed accuracy, by the law of large numbers. [1]

Estimate class-conditional probability

A naive (and *wrong*) estimator of class-conditional probability $P(\mathbf{x}|c)$ is the average of samples directly taken from training samples, i.e.

$$\widehat{P(\mathbf{x}|c)} = \frac{|D_{c,\mathbf{x}}|}{|D_c|}$$

where $D_{c,\mathbf{x}}$ is the training sample of label c and has value $\mathbf{x}$. However, this approach will not work. $|D_{\mathbf{x}}| = n^d$ when there are d attributes (dimensions) and n possible values for each attribute, which is further larger than the possible number of training samples when the dimension is high. It is easy to imagine the case where most of $\frac{|D_{c,\mathbf{x}}|}{|D_c|}$ is zero due to no sample is equal to the certain $\mathbf{x}$. [2]

Naive Bayesian considers $P(\mathbf{x}|c)$ in a different way. It assumes different attributes in $\mathbf{x}$, i.e. $x_0, x_1, ..., x_{d-1}$. Then the class-conditional can be expressed as

$$P(\mathbf{x}|c) = \prod_{i=0}^{d-1} P(x_i|c)$$

In this case, x_i will not have the exponential cardinality as $\mathbf{x}$, hence, it can be measured by the estimator

$$\widehat{P(x_i|c)} = \frac{|D_{c,x_i}|}{|D_c|}$$

for discrete attributes. And for continuous attributes, it can be measured under a Gaussian assumption $P(x_i|c) \sim$

$\mathcal{N}(\mu_{c,i}, \sigma_{c,i})$, and get the estimate by

$$\widehat{P(x_i|c)} = \frac{1}{\sqrt{2\pi}\widehat{\sigma_{c,i}}} \exp\left(-\frac{(x_i - \widehat{\mu_{c,i}})^2}{2(\widehat{\sigma_{c,i}})^2}\right)$$

where

$$\widehat{\sigma_{c,i}} = \frac{\sum_{i=0}^{n-1}(x_i - \widehat{\mu_{c,i}})^2}{n-1}$$

and

$$\widehat{\mu_{c,i}} = \frac{\sum_{i=0}^{n-1} x_i}{n}$$

which are the sample mean and sample standard deviation. [1]

Laplacian Correction

However, through the training samples are theoretically enough, but it is still possible that $\widehat{P(x_i|c)} = 0$ for some i. This pitfall will bring severe problem because if one $\widehat{P(x_i|c)}$ is zero, then the whole product will be zero; but in reality, it is not the case that one non-resembling attribute can eliminate the contribution of other attributes. Hence, naive Bayesian introduces **Laplacian correction** towards estimators for $P(c)$ and $P(x_i|c)$, where N_i is the number of possible values of $i-$th attribute:

$$\widehat{P(c)} = \frac{|D_c| + 1}{|D| + |\mathcal{Y}|}$$
$$\widehat{P(x_i|c)} = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}$$

which introduces an extra probability to eliminate zeros. This extra probability will not influence the result, if a prior which assumes a) labels are uniformly distributed and b) attributes are uniformly distributed on every label are uniformly distributed. [2]

Conclusion and Pseudo-code

After the estimation of the prior $P(c)$ and class-conditional $P(\mathbf{x}|c)$, naive Bayesian can give the classification

$$h_{nb}(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} \widehat{P(c)} \prod_{i=0}^{d-1} \widehat{p(x_i|c)}$$

The pseudo-code is shown in Algorithm 2.

References.

[1] Horst Hohberger. *VE401 – Probabilistic Methods in Engineering (lecture slides)*. 2021 (cit. on pp. 4, 5).

[2] Zhou Zhihua. *Machine Learning*. Tsinghua University Press, 2016 (cit. on pp. 4, 5).

Boltzmann Machines (restricted, deep)

- *Algorithm:* Boltzmann Machines (restricted, deep) (algo. 3-4)
- *Input:* training samples
- *Complexity:* not relevant for this topic

Algorithm 2: Naive Bayesian Classification

Input : training set D, the labels $\mathcal{Y}$, the testing sample $\mathbf{x}$
Output: The predicted label *label*

foreach $c \in \mathcal{Y}$ **do**
  $|D| \leftarrow$ the cardinality of D;
  $|Y| \leftarrow$ the cardinality of Y;
  $|D_c| \leftarrow 0$;
  foreach $i \in [0, d-1]$ **do**
    $N[i] \leftarrow$ the number of possible values of i-th attribute;
    $D_{c,x[i]} \leftarrow 0$;
  end foreach
  foreach $data \in D$ **do**
    $|D| \leftarrow |D| + 1$;
    if $data.label = c$ **then**
      $|D_c| \leftarrow |D_c| + 1$;
      if $data[i] = x[i]$ **then**
        $D_{c,x[i]} \leftarrow D_{c,x[i]} + 1$;
      end if
    end if
  end foreach
  $\widehat{P(c)} \leftarrow (|D_c| + 1)/(|D| + |\mathcal{Y}|)$;
  $\widehat{P(x[i]|c)} = (|D_{c,x[i]}| + 1)/(|D_c| + N[i])$;
  $prob[c] \leftarrow \widehat{P(c)} \times \prod_{i=0}^{d-1} \widehat{P(x[i]|c)}$;
end foreach
$label \leftarrow \arg\max_{c \in \mathcal{Y}} prob[c]$;
return *label*;

- *Data structure compatibility:* bipartite graph
- *Common applications:* neural networks

Problem. Boltzmann Machines (restricted, deep)

Improve the traditional restricted Boltzmann machine by adding more hidden layers. The previous method use directed edges to connect hidden layers, however, this is not the natural expansion of Boltzmann machine and does not consider the bidirectional propagation. Hence, a deep network with bidirectional connection between restricted Boltzmann machine needs to be developed.

Description

The original learning method for Boltzmann machine [3], using the stochastic gradient ascend to train, is too slow to train even with simulated annealing, since its complete graph nature needs heavy calculation. A more improved version is the restricted Boltzmann machine (RBM), which has no connections within hidden layer and visible layer [1]. RBM can be trained by a better method called contrastive divergence (CD).

Moreover, with the deep learning introduced, one hopes several RBMs can be stacked together, in order to pre-train RBMs layer by layer to speed-up the training of the whole deep neural network. If it is restricted to be only singly propagating from bottom layer to top layer, it can be modified as a deep belief network (DBN) [2], which is a stack of RBMs. To train DBN, one needs to pre-train each layer of RBM to get a faster convergence speed. However, DBN is not Boltzmann machine, since RBM uses non-directed edges but DBN uses directed. Hence, it only considers one direction of the probability propagation. It needs to be improved to a true deep Boltzmann machine (DBM) to get a more accurate pre-training.

In order to get the bidirectional conditional probability to pre-train a DBM, one may take the mean of the

conditional probability coming from upper layer and lower layer. This can solve part of the problem, however, it will bring so-called "double-counting" problem: the top layer and bottom layer will be calculated twice.

To solve this problem, DBM modifies the first and last RBM in the stack: for the bottom RBM, double the visible layer; for the top RBM, double the top hidden layer. In that way, the double-counting problem will be eliminated, and DBM can be pre-trained. [4]

Restricted Boltzmann Machine

Restricted Boltzmann machine is a neural network with one visible layer and one hidden layer. Like the general Boltzmann machine, its learning goal is to maximize the log-likelihood of one certain training sample, which is the input of RBM. Restricted Boltzmann machine has a faster training speed than general Boltzmann machine. This can be attributed to the bipartite characteristic of RBM, that is, "restrict" the Boltzmann machine to not have connections within both visible layer and hidden layer. This difference can be seen in Figure 1.

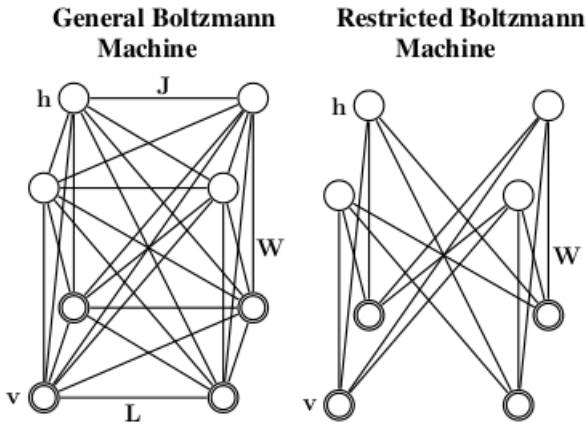

Figure 1: From general Boltzmann machine to restricted Boltzmann machine [4]

RBM often uses **contrastive divergence** (CD) algorithm to maximize the log-likelihood, which is also faster than the stochastic gradient ascend (SGA) method used for general Boltzmann machine. Contrastive divergence will be introduced in the next section.

Contrastive Divergence

Contrastive divergence algorithm is used as an improvement of the gradient ascend of log-likelihood of the training sample probability in restricted Boltzmann machine. [1]

Denote the visible vector as $\mathbf{v}$ and hidden layer as $\mathbf{h}$, we has the conditional probability

$$P(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{d} P(v_i|\mathbf{h})$$
$$P(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{q} P(h_j|\mathbf{v})$$

CD algorithm uses $\mathbf{v}$ calculate the probability distribution of hidden layer, and then samples to get an estimate of $\mathbf{h}$. Then use $\mathbf{h}$ to estimate $\mathbf{v}'$, and then $\mathbf{h}'$, use these four qualities to update the connection weight. [5] The algorithm is shown in Algorithm 3. [4]

Algorithm 3: Contrastive Divergence

Input : visible layer vector v
Output: the trained restricted Boltzmann machine

Function `sampling`$(P(\mathbf{m}|\mathbf{n}), \mathbf{n})$**:**
  foreach *dimension* m_i *of* $\mathbf{m}$ **do**
    $r \leftarrow$ random number from 0 to 1;
    if $r < P(m_i = 1|\mathbf{n})$ **then**
      $m_i \leftarrow 1$;
    else
      $m_i \leftarrow 0$;
    end if
  end foreach
  return $\mathbf{m}$;
end

$max \leftarrow$ maximum iteration number;
$w \leftarrow$ initial value of weight matrix;
for $i = 0$ **to** max **do**
  $\mathbf{h} \leftarrow$ `sampling`$(\prod_{j=1}^{q} P(h_j|\mathbf{v}), v)$;
  $\mathbf{v}' \leftarrow$ `sampling`$(\prod_{i=1}^{d} P(v_i|\mathbf{h}), h)$;
  $\mathbf{h}' \leftarrow$ `sampling`$(\prod_{j=1}^{q} P(h_j|\mathbf{v}'), v')$;
  $w \leftarrow w + \eta(\mathbf{v}\mathbf{h}^\top - \mathbf{v}'\mathbf{h}'^\top)$;
end for
return

Pre-training of Deep Restricted Boltzmann Machine

With the deep learning introduced, one hopes several RBMs can be stacked together, composing a deep network with plural hidden layers. If it is restricted to be only singly propagating from bottom layer to top layer, it is modified as a **deep belief network** (DBN), since the directed propagating posterior is of the style of Bayesian network. However, one may want to consider the top-down feedback, in order to allow better propagate uncertainty and deal with more robustly with ambiguous inputs [4]. Thus, **deep Boltzmann machine** (DBM) is introduced. The difference between DBN and DBM is shown in Figure 2.

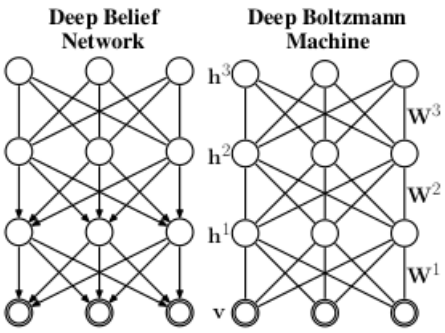

Figure 2: Difference between deep belief network and deep Boltzmann machine [4]

In order to speed-up the training speed in DBM, one prefers to do a layerwise pre-training. Layerwise pre-training means training each RBM layer by layer, from the bottom to the top. In this process, take the output layer of the current RBM as the input layer of the next RBM. Take an example of a deep Boltzmann machine composed of two restricted Boltzmann machine (see in Figure 3). The visible layer of the upper RBM is taken as the hidden layer of the lower RBM.

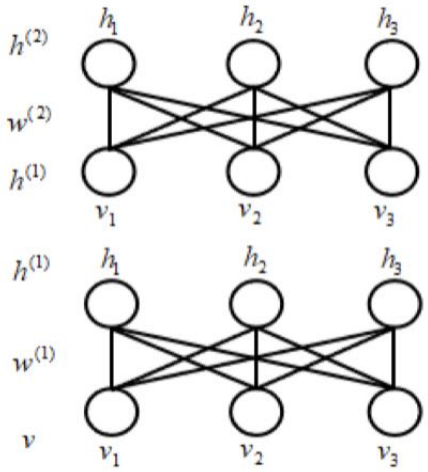

Figure 3: A two layer deep Boltzmann machine

First, by Algorithm 3, learn the weight of lower RBM. In the lower RBM, we have

$$\begin{aligned} P(\mathbf{v}) &= \sum_{\mathbf{h^{(1)}}} P(\mathbf{h^{(1)}}, \mathbf{v}) \\ &= \sum_{\mathbf{h^{(1)}}} P(\mathbf{h^{(1)}}) P(\mathbf{v}|\mathbf{h^{(1)}}) \\ &= \sum_{\mathbf{h^{(1)}}} P(\mathbf{h^{(1)}}; w^{(1)}) P(v|\mathbf{h^{(1)}}; w^{(1)}) \end{aligned}$$

which can be used to represent the lower RBM.

After that, one can use the similar sampling method decribed in Algorithm 3, to get an estimate of $\mathbf{h^{(1)}}$. Now it is easy to train the upper restricted Boltzmann machine in the same way as before.

However, in the upper RBM, we have

$$P(\mathbf{h^{(1)}}; w^{(2)}) = \sum_{\mathbf{h^{(2)}}} P(\mathbf{h^{(1)}}, \mathbf{h^{(2)}}; w^{(2)})$$

where the posterior $P(\mathbf{h^{(1)}}; w^{(2)})$ of $\mathbf{h^{(1)}}$ is derived by the argument (i.e. weight) matrix $w^{(2)}$, but not $w^{(1)}$ as that in lower RBM. In reality, since deep Boltzmann machine propagate bidirectionally, the posterior should be $P(\mathbf{h^{(1)}}; w^{(1)}, w^{(2)})$. Therefore, it should be reasonable to estimate the true posterior by a mean value of both $P(\mathbf{h^{(1)}}; w^{(1)})$ and $P(\mathbf{h^{(1)}}; w^{(2)})$, that is, propagating $w^{(1)}/2$ bottom-up and $w^{(2)}/2$ top-down [4]. In this method, the posterior $P(\mathbf{h^{(1)}}; w^{(1)}, w^{(2)})$ can be calculated as **aggregated posterior**

$$\begin{aligned} &\sum_{\mathbf{v}} P(\mathbf{h^{(1)}}|\mathbf{v}; w^{(1)}) + \sum_{\mathbf{h^{(2)}}} P(\mathbf{h^{(2)}}|\mathbf{h^{(1)}}; w^{(2)}) \\ =&\frac{1}{N} \sum_{\mathbf{v} \in V} P(\mathbf{h^{(1)}}|\mathbf{v}; w^{(1)}) + \frac{1}{N} \sum_{\mathbf{h^{(2)}} \in H} P(\mathbf{h^{(2)}}|\mathbf{h^{(1)}}; w^{(2)}) \end{aligned}$$

However, this will bring a problem, which is called **double-counting**. In order to get an estimate of the real posterior, the hybrid of $P(\mathbf{h^{(1)}}; w^{(1)})$ and $P(\mathbf{h^{(1)}}; w^{(2)})$ is used. However, in this sampling process, both $\mathbf{h^{(1)}}$ and $\mathbf{h^{(2)}}$ utilize the input vector $\mathbf{v}$, which will amplify the error if the input vector contains noise [4].

A small change is introduced to solve double-counting problem: for the top RBM, double the output hidden layer and link both of them to the input hidden layer; for the bottom RBM, double the input visible layer and link both of them to the output hidden layer. The modification process is illustrated in Figure 4. In this example, for the previously suspicious v, $2\mathbf{W^1}$ is propagated bottom-up but $\mathbf{W^1}$ is propagated top-down, hence,

the double-counting effect is eliminated.

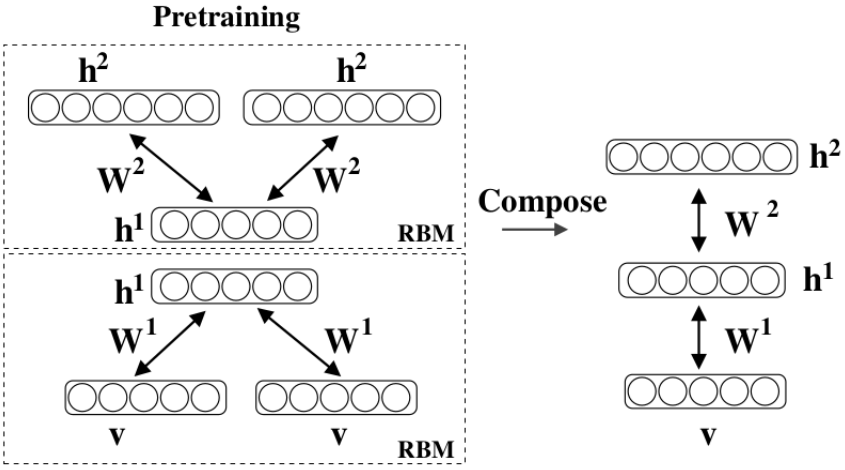

Figure 4: The solution to double-counting problem

The pseudo-code for the pre-training is in Algorithm 4. After pre-training, run contrastive divergence or stochastic gradient ascend for the whole network, one can get good result with a reasonable time cost.

Algorithm 4: Pre-training of DBM

Input : visible layer vector v, untrained DBM
Output: the pre-trained DBM

$max \leftarrow$ maximum iteration number;
for $c \leftarrow 0$ **to** $DBM.size - 1$ **do**
  $RBM \leftarrow DBM[c]$; `/* the i-th RBM from bottom to the top */`
  $w \leftarrow$ initial value of weight matrix for RBM;
  if $c = 0$ **then**
    $\mathbf{h_1} \leftarrow \mathbf{v}$;
  else
    $\mathbf{h_1} \leftarrow$ upper layer of $DBM[c-1]$;
  end if
  $\mathbf{h_2} \leftarrow$ upper layer of RBM;
  for $k = 0$ **to** max **do**
    if $c = 0$ **then**
      $p(h_{2,j} = 1|\mathbf{h_1}) \leftarrow \sigma(2\sum_i w_{ij}\mathbf{h_{1,i}})$;
    else
      $p(h_{2,j} = 1|\mathbf{h_1}) \leftarrow \sigma(\sum_i w_{ij}\mathbf{h_{1,i}})$;
    end if
    if $c = DBM.size - 1$ **then**
      $p(h_{1,i} = 1|\mathbf{h_2}) \leftarrow \sigma(2\sum_j w_{ij}\mathbf{h_{2,j}})$;
    else
      $p(h_{1,i} = 1|\mathbf{h_2}) \leftarrow \sigma(\sum_j w_{ij}\mathbf{h_{2,j}})$;
    end if
    $\mathbf{h_2} \leftarrow \mathtt{sampling}(p(h_{2,j} = 1|\mathbf{h_1}), \mathbf{h_1})$;
    $\mathbf{h'_1} \leftarrow \mathtt{sampling}(p(h_{1,i} = 1|\mathbf{h_2}), \mathbf{h_2})$;
    $\mathbf{h'_2} \leftarrow \mathtt{sampling}(p(h_{2,j} = 1|\mathbf{h_1}), \mathbf{h'_1})$;
    $w \leftarrow w + \eta(\mathbf{h_1}\mathbf{h_2}^\top - \mathbf{h'_1}\mathbf{h'_2}^\top)$;
  end for
end for
return

References.

[1] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8 (2002), pp. 1771–1800 (cit. on pp. 6, 7).

[2] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554 (cit. on p. 6).

[3] Geoffrey E Hinton and Terrence J Sejnowski. "Optimal perceptual inference". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. Vol. 448. Citeseer. 1983 (cit. on p. 6).

[4] Ruslan Salakhutdinov and Geoffrey Hinton. "Deep boltzmann machines". In: *Artificial intelligence and statistics*. PMLR. 2009, pp. 448–455 (cit. on pp. 7–9).

[5] Zhou Zhihua. *Machine Learning*. Tsinghua University Press, 2016 (cit. on p. 7).

TRPO (Trust Region Policy Optimization)

- *Algorithm:* TRPO (algo. 5)
- *Input:* training samples
- *Complexity:* not relevant for this topic
- *Data structure compatibility:* None
- *Common applications:* policy optimization in Reinforcement Learning and

Problem. TRPO (Trust Region Policy Optimization)

Given reinforcement learning training sample, optimize the process of policy gradient method and make it monotonically converge to the optimized policy.

Description

In the area of reinforcement learning, the are value-based methods policy-based methods. And the policy-based method can be further divided into three different major streams

1. **The policy iteration method**, which iterating between evaluating the value function under current policy and improving the policy [1]. This is almost the most classic method for policy optimization.
2. **The policy gradient method**, which makes the policy function increase along the gradient to reach the local maximum [3].
3. **The derivative-free methods**.

The policy-based have some advantages compared with the value-based methods, including: [5]

1. The policy-based methods are always improved towards the optimized result in each iteration, although each time a small amount. But the value-based methods may oscillate around the optimized result without hitting the optimized point. In this sense, the policy-based methods usually has a better convergence performance.
2. When the dimension of the space becomes high enough, the value-based methods need to compare lots of values resulting from choosing different actions in order to generate a optimized action. Thus, the policy-based methods have a more efficient approach in these situations.
3. The value-based methods can seldom learn some random policy but the policy-based methods can. And there exists some models where the random policy is the optimal one. A trivial example is playing the

rock-paper-scissors game. A predefined strategy may be discovered by enemy while a randomized strategy cannot be predicted.

4. Calculating the value function may be complicated in some cases. For example, when one want to catch a ball falling from the sky, it is hard to calculate the corresponding value for a certain action given the position of the ball. But the policy-based methods can easily handle this situation by simply moving towards the falling direction of the ball.

Meanwhile, there are also shortcomings of the policy-based methods. The naive policy-based method like the policy gradient method may not be as efficient as the value-based ones, since the policy-based method that will only move towards the policy gradient a little bit each time. While the value-based methods will always push the agent towards the action with the largest value or reward. Also, for the policy gradient-based optimization, the equation to update the new policy is

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \nabla_\theta J$$

A fatal problem of of policy-gradient method revealed in this equation is that, the step length α should be chosen wisely. Otherwise, one cannot ensure that the new policy updated using this equation is no worse than the older one. The consequence of having a bad step length is the learning process will be come worse each iteration and will reach a crash state finally.

Under such circumstance, TRPO is designed to solve:

1. Since gradient-based methods enjoy much better sample complexity guarantees than gradient-free methods, extending their success to reinforcement learning would allow for efficient training of complex and powerful policies [4].

2. Find new functions to decide the step length, ensuring a non-decreasing update of the rewarding function.

In order to construct a model to ensure the rewarding function to be non-decreasing, an equation built by Kakade [2] is adopted as the base of development:

$$\eta(\tilde{\pi}) = \eta(\pi) + E_{s_0, a_0, \ldots \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi \left(s_t, a_t \right) \right]$$

the η represents the cost of the policy, the $\tilde{\pi}$ represents the new policy and π represents the original policy or old policy. Also note that

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

is the advantage function, which essentially represents whether this action receives better reward compared with average value. Thus, by ensuring the value of advantage function to be non-negative, one can equivalently ensure the new policy is no-worse than the previous one.

Then, the difference of the cost can be further written in the form

$$\sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a \mid s) A_\pi(s, a)$$

This is because the expectation of the advantage function can be written as

$$\sum_{t=0}^{\infty} \sum_s P\left(s_t = s \mid \tilde{\pi}\right) \sum_a \tilde{\pi}(a \mid s) \gamma^t A_\pi(s, a)$$

Note that the

$$P\left(s_t = s \mid \tilde{\pi}\right) \tilde{\pi}(a \mid s)$$

represents that the possibility of doing action a with the condition of state s and the strategy of $\tilde{\pi}$. Also define

the discounted visitation frequencies to be

$$\rho_\pi(s) = \left(P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + ...\right)$$

With that, the relationship between the cost of new policy and old policy can be further written as

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a \mid s) A_\pi(s, a)$$

One problem of the equation discussed above is that the advantage function part is still relying on the new strategy $\tilde{\pi}$. The author then argues the possibility and rationality of replacing the new policy with the old policy on the right hand side of the equation, namely

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a \mid s) A_\pi(s, a)$$

This is an approximation with the precondition of the difference of parameter of π being small. Furthermore, the importance sampling to second term of the replacement function $L_\pi(\tilde{\pi})$, it becomes

$$L_\pi(\tilde{\pi}) = \eta(\pi) + E_{s \sim \rho_{\theta_{\text{old}}}, a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\tilde{\pi}_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a|s)} A_{\theta_{\text{old}}}(s, a) \right]$$

In the paper, the author provides the proof that the newly constructed $L_\pi(\tilde{\pi})$ and the original $\eta(\tilde{\pi})$ match in the first order. That result can be written as

$$\begin{aligned} L_{\pi_{\theta_{\text{old}}}}(\pi_{\theta_{\text{old}}}) &= \eta(\pi_{\theta_{\text{old}}}) \\ \nabla_\theta L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\Big|_{\theta=\theta_{\text{old}}} &= \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_{\text{old}}} \end{aligned}$$

Thus, in the neighbor of θ_{old}, optimization of the function L will also result in optimization of the original reward function η. The remaining question is how to determine the step length of the improvement.

It can be proved that [4]

$$\eta(\tilde{\pi}) \geqslant L_\pi(\tilde{\pi}) - C D_{KL}^{\max}(\pi, \tilde{\pi})$$
$$\text{where } C = \frac{2\varepsilon\gamma}{(1-\gamma)^2}$$

And then the problem becomes

$$\begin{aligned} &\text{maximize}_\theta \; E_{s \sim \rho_{\theta_{\text{old}}}, a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A_{\theta_{\text{old}}}(s, a) \right] \\ &\textit{subject to} \\ &D_{KL}^{\max}(\theta_{\text{old}}, \theta) \leq \delta \end{aligned}$$

Furthermore, replace D_{KL}^{max} with $\bar{D}_{KL}^{\theta_{old}}$ and $\rho_{theta_{old}}$ with $\pi_{\theta_{old}}$, the problem is finally reduced to

$$\begin{aligned} &\text{maximize}_\theta \; E_{s \sim \pi_{\theta_{\text{old}}}, a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A_{\theta_{\text{old}}}(s, a) \right] \\ &\text{subject to } E_{s \sim \pi_{\theta_{\text{old}}}} \left[D_{KL}\left(\pi_{\theta_{\text{oid}}}(\cdot|s) \| \pi_\theta(\cdot|s)\right) \right] \leq \delta \end{aligned}$$

The algorithm for TRPO is shown in algorithm 5

Algorithm 5: TRPO

Input : initial policy parameters θ_0, initial value function parameter ϕ_0
Output: The optimized policy

for $k \leftarrow 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment;

Compute rewards-to-go $\hat{R}_s$;

Compute advantage estimation, $\hat{A}_t$ based on the current value function V_{ϕ_0};

Estimate policy gradient as

$$\hat{g_k} \leftarrow \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t;$$

Use the conjugate gradient algorithm to compute

$$\hat{x_k} \approx \hat{H_k}^{-1} \hat{g_k};$$

Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x_k}^T \hat{H_k} \hat{x_k}}} \hat{x_k};$$

where $j \in \{1, 2, 3, ..., K\}$ is the smallest value which improves the sample loss and satisfy the sample KL-divergence constraint.

Fit the value function by regression on mean-square error

$$\phi_{k+1} = \mathit{argmin}_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

typically via some gradient descent algorithm;

end for

return

References.

[1] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*. Vol. 1. Athena scientific, 2012 (cit. on p. 11).

[2] Sham Kakade and John Langford. "Approximately optimal approximate reinforcement learning". In: *In Proc. 19th International Conference on Machine Learning*. Citeseer. 2002 (cit. on p. 12).

[3] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697 (cit. on p. 11).

[4] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897 (cit. on pp. 12, 13).

[5] David Silver. *COMPM050/COMPGI13 Reinforcement Learning, UCL*. 2021 (cit. on p. 11).

Lens Distortion

- *Algorithm:* Lens Distortion Represent (algo. 6) and Correct (algo. 7)
- *Input:* A picture P
- *Complexity:* not relevant for this topic
- *Data structure compatibility:* None

- *Common applications:* camera calibration, post production of pictures

Problem. Lens Distortion

Given a picture that is distorted (only the central point is not distorted in worst case), fix the distortion and output the picture containing objects in plain-like vision.

Description

One of the major problem faced by all the photographers is the distortion of the pictures taken after adding the zoom lenses, in particular, the large-range zoom lenses. Yet there is still possibility that even pictures shot by cameras using the prime lenses will meet some distortions. This kind of distortion is defined as the **lens distortion** in photography. Lens distortion is a deviation from the ideal projection considered in pinhole camera model. It is a form of optical aberration in which straight lines in the scene do not remain straight in an image [3]. According to the distortion pattern, the lens distortion is divided into **tangential distortion and radial distortion** based on the direction of the distortion, as shown in figure 5. Particularly, there are two kinds of

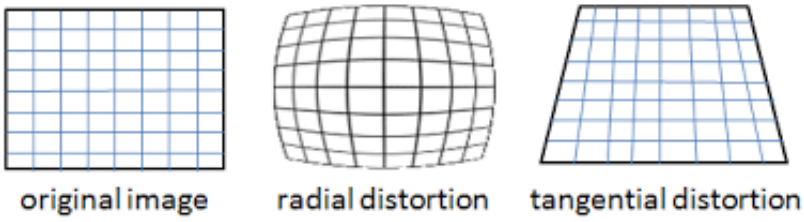

Figure 5: Classification of Distortion

radial distortions and they are named after the shape of the distortion, namely, **the Barrel distortion and the Pincushion distortion**. The shapes of the distortions can be seen in the figure 6. In some extreme cases,

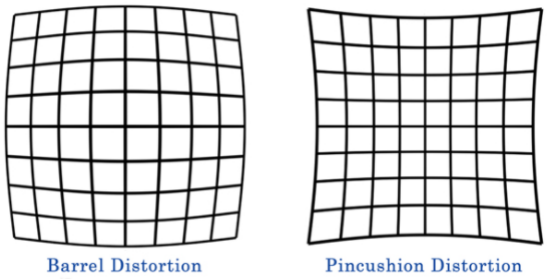

Figure 6: Two Traditional kinds of Radial Distortions

the lenses of the camera are arranged so poorly that both the Barrel Distortion and the Pincushion distortion happen at the same time. The lines near the central point experience a Barrel Distortion while the lines near the corners experience a Pincushion Distortion. This kind of distortion is called **Mustache Distortion** and is shown in figure 7.

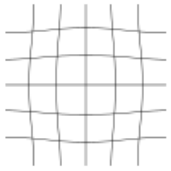

Figure 7: The Mustache Distortion

Calibration Models Solving Lens Distortion

To solve lens distortion problem, in the year of 1919, A. Conrady first introduced the **Decentering Distortion Model**. And scholars built the **Thin Prism Model** later. After that, one of the popular solution to solve the lens distortion was introduce by Brown in [2], which is then called **Brown-Conrady Model**. There are more models built by researchers recently and one of them is developed by the Institute of Image Processing and Pattern Recognition of SJTU [4].

a) Thin Prism Model

In his paper, Brown first pointed out that the **Thin Prism Model** could be described using the mathematical equations

$$\Delta_r(x, y) = P\cos(\phi - \phi_0)$$
$$\Delta_t(x, y) = P\sin(\phi - \phi_0)$$

where $\Delta_r(x, y)$ is the radial component of the lens distortion at plate of coordinate (x, y) and $\Delta_t(x, y)$ is the tangential component of the lens distortion at plate of coordinate (x, y). The other letters have the relation as follows:

$$\begin{aligned}
P =& c\left[(\cos\theta_1\cos\mu\epsilon - \cos\theta_3)\sin\mu\epsilon\right. \\
&\left.+(1-\cos\epsilon\cos\mu\epsilon)\sin\theta_0\sin(\phi-\phi_0)\right] \\
r =& \left(x^2+y^2\right)^{1/2} \\
\sin\theta_0 =& r/\left(r^2+c^2\right)^{1/2} \\
\sin\phi =& x/r \\
\cos\phi =& y/r \\
\sin\theta_1 =& (\sin\theta_0)/\mu \\
\cos\theta_2 =& \sin(\phi-\phi_0)\sin\theta_1\sin\epsilon + \cos\theta_1\cos\epsilon \\
\sin\theta_3 =& \mu\sin\theta_2.
\end{aligned}$$

where μ is the index of the refraction of the glass,c is the principle distance, and ϵ is a free parameter greater than 0. Also, ϕ_0 is defined as the angle between the positive x-axis and the image of edge of prism.

We are now able to describe the algorithm to calculate the lens distortion of a point as algorithm 6.

Algorithm 6: Get the lens distortion using Thin Prism Model

Input : point P with coordinate (x, y), μ, ϵ, c, ϕ_0
Output: The radial and tangential lens distortion Δ_r and Δ_t for P

$r \leftarrow \sqrt{x^2 + y^2}$;
$\theta_0 \leftarrow \arcsin \frac{r}{\sqrt{r^2+c^2}}$;
$\phi \leftarrow \arcsin \frac{x}{r}$;
$\theta_1 \leftarrow \arcsin \frac{\sin \theta_0}{\mu}$;
$\theta_2 \leftarrow \arccos\left(\sin\left(\phi - \phi_0\right) \sin\theta_1 \sin\epsilon + \cos\theta_1 \cos\epsilon\right)$;
$\theta_3 \leftarrow \mu \sin\theta_2$;
$P \leftarrow c\left[\left(\cos\theta_1 \cos\mu\epsilon - \cos\theta_3\right) \sin\mu\epsilon + (1 - \cos\epsilon \cos\mu\epsilon) \sin\theta_0 \sin\left(\phi - \phi_0\right)\right]$;
$\Delta_r(x, y) \leftarrow P\cos(\phi - \phi_0)$;
$\Delta_t(x, y) = P\sin(\phi - \phi_0)$;
return $\{\Delta_r(x, y), \Delta_t(x, y)\}$

After representing the radial and tangential components of lens distortion at certain plate, some compensation methods can be applied to eliminate most of the distortion. But no method can guarantee to remove all the distortion of happened to the picture. In particular, the method of **Project Compensation** is introduced to the Thin Prism Model [2].

b) Projective Compensation

The project compensation is essentially a form of project transformation. And the projective transformation describes what happens to the perceived positions of observed objects when the point of view of the observer changes. A concrete example is shown in figure 8

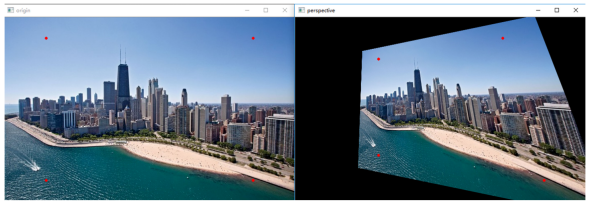

Figure 8: An example of Projective Transformation

The implementation for this transformation is multiplying the coordinate for any point $(x, y, z, 1)$ with a 4 by 4 transformation matrix

$$\begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [1]$$

And the do the tilt to the picture afterwards. Based on the information above, the projective compensation method can be demonstrated using the equations

$$\Delta_r(x, y) = \left(P + b - c\gamma - \frac{\gamma}{c} r^2\right) \sin\left(\phi - \phi_0\right)$$
$$\Delta_t(x, y) = \left(P + b - c\gamma\right) \cos\left(\phi - \phi_0\right)$$

where b and γ represent the magnitudes of compensate translation and tilt. These two equations are actually a concrete expression of the composition of the compensate matrix and the lens distortion at point (x, y).

Now, the algorithm of the compensation step can be written as 7

Algorithm 7: Projective Compensate for Thin Prism Model

Input : A compensate matrix M, a tilt matrix T and a point P with coordinate (x, y)
Output: The radial distortion Δ_r and tangential distortion Δ_t after compensation for P

1 $b, \gamma \leftarrow$ calculate the magnitude b, γ from the given matrix M, T (i.e. $b \leftarrow |MP|$ and $\gamma \leftarrow |TP|$);
2 $\Delta_r(x, y) \leftarrow (P + b - c\gamma - \frac{\gamma}{c}r^2)\sin(\phi - \phi_0)$;
3 $\Delta_t(x, y) \leftarrow (P + b - c\gamma)\cos(\phi - \phi_0)$
4 **return** $\{\Delta_r(x, y), \Delta_t(x, y)\}$

One experiment conducted by Brown shows that this compensation successfully reduce the radial distortion by 53.7% and the tangential distortion by 22.2%.

c) Decentering Distortion Model and Brown-Conrady Model

As a matter of fact, the **Decentering Distortion Model** introduced by Conrady has the similar math expression as the **Thin Prism Model**. The difference is that the radial distortion component in Decentering Distortion Model is

$$\Delta_r(x, y) = 3P \sin(\phi - \phi_0)$$

which is three times larger of compared with the Thin Prism Model. And all the calculation and algorithms are similar as discussed in 6 and 7.

While the Decentering Distortion Model is very similar to the Thin Prism Model, the **Brown-Conrady Model** gives a more efficient way to correct the lens distortion. The major idea is to decompose the coefficient P into the sum of power series. As indicated in the equation

$$P = J_1 r^2 + J_2 r^4 + J_3 r^6 + \cdots$$

Then introduce new parameters

$$\begin{aligned} P_1 &= -J_1 \sin\phi_0 \\ P_2 &= J_1 \cos\phi_0 \\ P_3 &= J_2/J_1 \\ P_4 &= J_3/J_1 \\ &\vdots \end{aligned}$$

When the distortion is not that intensive, it is sufficient to use two terms $J_1 r^2 + J_2 r^4$ to express the coefficient with high accuracy. For special cameras like the fish-eye lens, the third term may be adopted as well. The equation for the x, y component of lens distortion can then be represented in the form of [2]

$$\begin{aligned} \Delta_x(x, y) &= \left[P_1\left(r^2 + 2x^2\right) + 2P_2 xy\right]\left[1 + P_3 r^2 + P_4 r^4 + \cdots\right] \\ \Delta_y(x, y) &= \left[2P_1 xy + P_2\left(r^2 + 2y^2\right)\right]\left[1 + P_3 r^2 + P_4 r^4 + \cdots\right] \end{aligned}$$

The corresponding algorithm to compute these components are shown in the algorithm 8.

Algorithm 8: Calculate the distortion component after correction using Brown-Conrady Model

Input : A point P with coordinate (x, y), ϕ_0, c, μ, ϵ

Output: The x-axis distortion Δ_x and y-axis distortion Δ_y after correction for P

$r \leftarrow \sqrt{x^2 + y^2}$;
$\theta_0 \leftarrow \arcsin \frac{r}{\sqrt{r^2+c^2}}$;
$\phi \leftarrow \arcsin \frac{x}{r}$;
$\theta_1 \leftarrow \arcsin \frac{\sin \theta_0}{\mu}$;
$\theta_2 \leftarrow \arccos\left(\sin\left(\phi - \phi_0\right) \sin\theta_1 \sin\epsilon + \cos\theta_1 \cos\epsilon\right)$;
$\theta_3 \leftarrow \mu \sin\theta_2$;
$P \leftarrow c\left[\left(\cos\theta_1 \cos\mu\epsilon - \cos\theta_3\right) \sin\mu\epsilon + (1 - \cos\epsilon \cos\mu\epsilon) \sin\theta_0 \sin\left(\phi - \phi_0\right)\right]$;
foreach J_i **do**
    /* i may be 4 or 5 depending on 2 term or 3 terms needed for evaluation */
    $P \leftarrow \frac{P}{r^2}$;
    $J_i \leftarrow P \mod r^2$;
end foreach
$P_1 \leftarrow -J_1 \sin\phi_0$;
$P_2 \leftarrow J_1 \cos\phi_0$;
foreach P_i $(i \geq 3)$ **do**
    $P_i \leftarrow \frac{J_{i-1}}{J_1}$;
end foreach
$\Delta_x(x, y) \leftarrow [P_1(r^2 + 2x^2) + 2P_2xy][1 + P_3r^2 + P_4r^4 + \cdots]$;
$\Delta_y(x, y) \leftarrow [2P_1xy + P_2(r^2 + 2y^2)][1 + P_3r^2 + P_4r^4 + \cdots]$;
return $\{\Delta_x(x, y), \Delta_y(x, y)\}$

Again, an experiment was conducted on this Brown-Conrady Model and proved that this model is far more efficient than the Thin Prism Model.

d) New Calibration Model Introduced by SJTU's Lab

Regarding the models discussed above, as well as other models constructed later, Wang concluded that the radial distortion, decentering distortion and thin prism distortion are coupled with one another, because both decentering distortion and thin prism distortion have a contribution to radial component[4].

The steps of formulating the lens distortion models can be concluded briefly as

1. Do the perspective projecting process.

$$\begin{bmatrix} x & y & f \end{bmatrix}^{\mathsf{T}} = \frac{f}{Z_c} \begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}^{\mathsf{T}},$$

2. Consider the radial distortion (first order and second order only).

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \left[1 + k_1\left(x^2 + y^2\right) + k_2\left(x^2 + y^2\right)^2\right] \begin{bmatrix} x \\ y \end{bmatrix},$$

3. Consider the rotation of the image.

$$\begin{bmatrix} x_{di} & y_{di} & z_{di} \end{bmatrix}^{\mathsf{T}} = R_c \begin{bmatrix} x_d & y_d & 0 \end{bmatrix}^{\mathsf{T}}$$

$$R_c = R_\psi \cdot R_\theta$$

$R_{\psi,\theta}$ are the rotation matrix with respect to x-axis and y-axis.

$$R_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}; \quad R_\psi = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}$$

4. The real image point in the sersor plane can be calculated as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 \\ 0 & \frac{1}{dy} \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$$

References.

[1] Sheng Bin. *CS337, Computer Graphics (lecture slides), SJTU*. 2021 (cit. on p. 17).

[2] Dean Brown. "Decentering distortion of lenses". In: 1966 (cit. on pp. 16–18).

[3] Olgierd Stankiewicz, Gauthier Lafruit, and Marek Domański. "Multiview video: Acquisition, processing, compression, and virtual view rendering". In: *Academic Press Library in Signal Processing, Volume 6*. Elsevier, 2018, pp. 3–74 (cit. on p. 15).

[4] Jianhua Wang, Fanhuai Shi, Jing Zhang, and Yuncai Liu. "A new calibration model of camera lens distortion". In: *Pattern recognition* 41.2 (2008), pp. 607–615 (cit. on pp. 16, 19).