ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐẠI HỌC QUỐC GIA TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN KHOA HỌC MÁY TÍNH

NHẬN DẠNG
BÁO CÁO MÔN HỌC CUỐI KỲ

Sinh viên thực hiện: Nguyễn Thế Hoàng (MSSV: 2012 0090)
Giáo viên phụ trách: Lê Thanh Phong - Lê Thành Thái

BÁO CÁO CUỐI KỲ - NHẬN DẠNG
HỌC KỲ II - NĂM HỌC 2022 - 2023

A-1: Local binary pattern (LBP)

This section is based mainly on [6] and [7]. All formula, information, specifications, etc. are collected from those references.

a. Describe the specification of $\mathbf{LBP_{P,R}}$

Formula

Differ from the original 3×3 matrix computation in the original LBP, this generic formulation of $\mathrm{LBP_{P,R}}$can be used for any size of the neighborhood or any number of sampled points.

Given an 2-D image $I(x, y)$ (each pixel is located at an arbitary location (x, y)); g_c is the gray scale of an arbitary pixel locates at (x, y). The whole picture is converted to monochromatic mode in grayscale color, or:

$$I(x, y) = g_c$$

In a neighborhood area created by a circular with radius R, center at a pixel (x, y) and choose P sampling points, we call g_p as the gray scale of the p-th sampling point:

$$g_p = I(x_p, y_p), p = 0, 1, \dots P - 1$$

The position of sampling points is calculated as:

$$x_p = x + R\cos(2\pi\frac{p}{P}),$$
$$y_p = y - R\sin(2\pi\frac{p}{P})$$

Now, a picture can be presented as a local gray-scale distribution (i.e texture), characterized as the joint distribution of gray scale of $P + 1(P > 0)$ neighboring pixels. We use this as a feature vector for that picture.

$$T \approx t(s(g_0 - g_c), s(g_1 - g_c), \dots, s(g_{P-1} - g_c))$$

which $s(z)$ is defined as:

$$s(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

The generic formula of $\mathrm{LBP_{P,R}}$can be derived from above joint distribution:

$$\boxed{\mathrm{LBP_{P,R}}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c)2^p}$$

for an arbitary center pixel (x_c, y_c) with circular neightborhood has radius R and choose P sampling points.

The local gray-scale distribution T, or a feture vector S, can thus be described with a 2^P-bin discrete distribution of LBP code:

$$S = T \approx t(\mathrm{LBP_{P,R}}(x_c, y_x)$$

Advantages and disadvantages

This is described as the Table 1.

Advantages	Disadvantages
• Not affected by the monotonic gray scale changes (e.x. by illumination chages). • Simple and quick computation. • Prove that operator applied to a small neighborhood area can be usually adequate. • Can be applied to various tasks in computer vision, not just for texture problems.	• Affected by the rotation variance. • Tremendous number of code and distribution.

Bảng 1: Comparision of advantages and disadvantages of $\mathrm{LBP}_{\mathrm{P,R}}$

b. Describe the specification of $\mathbf{LBP}^{\mathbf{ri}}_{\mathbf{P,R}}$

Formula

$\mathrm{LBP}^{\mathrm{ri}}_{\mathrm{P,R}}$ can be rotational invariance, when be compared with the $\mathrm{LBP}_{\mathrm{P,R}}$. We achieved that by using the following operator:

$$\boxed{\mathrm{LBP}^{\mathrm{ri}}_{\mathrm{P,R}} = \min_i \mathrm{ROR}(\mathrm{LBP}_{\mathrm{P,R}}, i)}$$

which $\mathrm{ROR}(x, i)$ means doing bitwise right roration of bit sequence x by i steps. In other words, for an arbitary $\mathrm{LBP}_{\mathrm{P,R}}$, we do bitwise right roration until we get the minimum binary sequence. As a result, if we use $\mathrm{LBP}^{\mathrm{ri}}_{8,1}$, we only need 36 codes when compared with $\mathrm{LBP}_{8,1}$ which we need 256 codes.

c. Describe the specification of $\mathbf{LBP}^{\mathbf{riu2}}_{\mathbf{P,R}}$

Formula

A local binary pattern is called uniform if it has at most two circular 0-1 or 1-0 transitions. With the experiment, we found out that uniform patterns occur nearly 90% of all patterns in images because the edge or boundary is main component in an image and uniform patterns can capture abrut changes in intensity of texture. So it is useless to label each non-uniform patterns individually. Not like $\mathrm{LBP}_{\mathrm{P,R}}$, in $\mathrm{LBP}^{\mathrm{riu2}}_{\mathrm{P,R}}$ we map each uniform pattern to an identical label, and the rest non-uniform pattern is put in the same label. As a result, we only need *59* output labels if we use $\mathrm{LBP}^{\mathrm{riu2}}_{8,1}$.

In addition to the fact that we can use $\mathrm{LBP}^{\mathrm{ri}}_{\mathrm{P,R}}$ to change every bit sequence to the minimum form and achieve the rotation invariance, the number of output labels is greatly reduced. For each uniform pattern, we use $\min_i \mathrm{ROR}(\mathrm{LBP}_{\mathrm{P,R}}, i)$ to change it to the minimum form, so the final number of output label is reducede from 59 to 10 (in case $\mathrm{LBP}^{\mathrm{riu2}}_{8,1}$), as shown in the Picture **??**

Test

A-2. Principal Component Analysis(PCA) and Linear Discriminant Analysis (LDA)

This section is based mainly on [9]. All formula, information, specifications, etc. are collected from those references. Any other references beside those will be cited in-text directly.

a. Describe the PCA

Define the problem

Given a dataset $\boldsymbol{X}$, which is a matrix $m \times n$, where m is the number of dimension of original feature, n is the number of samples. We want to find another basis, which is a linear combination of original basis that can best re-express the dataset $\boldsymbol{X}$. In term of "best re-express", we mean that the new representation of the dataset (we call it $\boldsymbol{Y}$) need to have low noise and remove the redundancy dimension features.

In other words, we need to find a matrix P that can transform $\boldsymbol{X}$ into $\boldsymbol{Y}$:

$$\boldsymbol{PX} = \boldsymbol{Y}$$

besides that:

- $\boldsymbol{p_i}$ are the rows of $\boldsymbol{P}$. $\{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_m}\}$ are a set of new basis vectors, or *principal component*.
- $\boldsymbol{x_i}$ are the columns of $\boldsymbol{X}$. Each $\boldsymbol{x_i}$ is an original sample.
- $\boldsymbol{y_i}$ are the columns of $\boldsymbol{Y}$. Each $\boldsymbol{y_i}$ is a new way to express respective $\boldsymbol{x_i}$.

We also need some assumptions of the dataset to set up a solution for PCA. That is, how a new $\boldsymbol{Y}$ can be seen as having low noise and non-redundancy?

Assumptions

- *Linearity*. It simplifies the problem by restristing the set of possible basis. As a result, PCA can re-expressing the data as a linear comination of some basis vectors.
- *Large variances have important structure*. We define the *covariance matrix* $\boldsymbol{C_Y}$ as:

  $$\boldsymbol{C_Y} = \frac{1}{n}\boldsymbol{Y}\boldsymbol{Y}^T$$

  This covariance matrix shows us if $\boldsymbol{Y}$ is expressed well enough (in terms of low noise and low redundancy). In $\boldsymbol{C_Y}$:

  - The diagonal terms are the *variance* of a particular feature. Large values *may* correspond to an interesting feature, or useful *signal*. On the other hand, low values *may* correspond to noise feature. We need to rotate the original basis such that it is parallel to the direction with largest variances.

  - The off-diagonal terms are the *covariance* between features. Large values *may* correspond to high redundancy.

    With this assumption, all off-diagonal of $\boldsymbol{C_Y}$ need to be 0s (to remove all redundancy). So, $\boldsymbol{C_Y}$ must be a diagonal matrix.

- *The principal components are orthogonal.* This will be used to show that we can use eigenvectors of $\boldsymbol{C_X}$ as *principal components* of $\boldsymbol{X}$.

Solving PCA using eigenvector decomposition

Using above observations, folloing are steps for solving PCA using eigenvector decomposition:

1. For each feature, compute means over all the samples. Call it $\overline{\boldsymbol{x}}$. For each sample $\boldsymbol{x_i}$, subtracting off the mean of each feature:

$$\boldsymbol{x_i} = \boldsymbol{x_i} - \overline{\boldsymbol{x}}$$

2. Compute the covariance of $\boldsymbol{X}$:

$$\boldsymbol{C_X} = \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^T$$

3. Compure the eigenvalues and eigenvectors of $\boldsymbol{C_X}$. We get $\boldsymbol{P}$ where each row $\boldsymbol{p_i}$ of $\boldsymbol{P}$ is an eigenvector of $\boldsymbol{C_X}$. Each $\boldsymbol{p_i}$ also has a respective eigenvector λ_i, which is also the variance of $\boldsymbol{X}$ along $\boldsymbol{p_i}$.

4. Sort the eigenvectors in $\boldsymbol{P}$ according to λ (or also respective variance). Decide the number of features we want to keep in the new basis (example we choose $d \ll m$). We receive a shortened matrix $\boldsymbol{P_d}$.

5. Project $\boldsymbol{X}$ to the new basis $\boldsymbol{P_d}$.

$$\boldsymbol{Y} = \boldsymbol{P_d}\boldsymbol{X}$$

c. Compare PCA with LDA

This is described as the Table 2.

A-3. Support Vector Machine

This section is based mainly on [1]. All formula, information, specifications, etc. are collected from those references. Any other references beside those will be cited in-text directly.

PCA	LDA
• Is an unsupervised technique. It ignores the label of the dataset. • Is often only used in dimensionality reduction. • Finds the directions have maximal variance values. • Outperform LDA when the number of samples per class is small or when the training data nonuniformly sample underlying distribution. [5]	• Is a supervised technique. • Can be used for both dimensionality reduction and classification tasks. • Find a feature subspace that maximise class seperability. • Make assumption that samples are normally distributed in each class and have equal class covariances. [5]

Bảng 2: Main differences between PCA and LDA technique

a. Describe the SVM

Following are major steps to build and train a SVM. Some formula derivation is described more clearly while some others are omitted.

Step 1 *Define basic linear classifier model*

Given a training dataset $\mathcal{D} = \{(\boldsymbol{x_i}, y_i) \mid \boldsymbol{x_i} \in \mathcal{R}^n, y_i \in \{-1, 1\}\}_{i=1}^m$. Find $\boldsymbol{w} = \{w_0, w_1, \ldots, w_n\}$ such that $\text{sign}(\boldsymbol{w} \cdot \boldsymbol{x_i}) = y_i$ for every $\boldsymbol{x_i}$.

Here, m is the number of training samples, n is the number of dimension of features. $\boldsymbol{x_i}$ is the i-th training instance. y_i is the label of respective $\boldsymbol{x_i}$. We need to find a value of $\boldsymbol{w}$ which represent a hyperplanes seperates classes, like the Image **??**.

Step 2 *Define geometric margin for comparing optimal hyperplane*

A pure searching hyperplane may end up different solutions for each time we train the model. The goal of SVM is always returning the most *optimal hyperplane* that have maximum distance to the nearest instance of each class.

Given a dataset $\mathcal{D}$, for a hyperplane we compute its *functional margin* F:

$$F = \min_{i=1,\ldots,m} y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)$$

(which can be expressed as the smallest distance between the hyperplane and a training instance, but also need to classify correctly).

To avoid always finding the best hyperplane just by rescaling $\boldsymbol{w}$ and b, we will use the *geometric margin* M instead to compare hyperplanes.

$$\boxed{M = \min_{i=1,\ldots,m} y_i \left(\frac{\boldsymbol{w}}{\| \boldsymbol{w} \|} \cdot \boldsymbol{x_i} + \frac{b}{\| \boldsymbol{w} \|} \right) = \min_{i=1,\ldots,m} y_i \left(\gamma_i \right)}$$

Step 3 *Define the SVM optimization problem*

Now, the goal of SVM is *finding the optimal seperating hyperplane which is defined by the normal vector* $\boldsymbol{w}$ *and the bias* b *such that the geometric margin* M *is the largest.* In other word, SVM trys to solve below optimiation problem:

$$\max_{\boldsymbol{w},b} \; M$$
$$\text{s.t. } \gamma_i \geq M, \; i = 1, \ldots, m$$

As derived in [<**empty citation**>], the above problem can be re-express as:

$$\min_{\boldsymbol{w},b} \; \frac{1}{2} \parallel \boldsymbol{w} \parallel^2$$
$$\text{s.t. } y_i(\boldsymbol{w} \cdot \boldsymbol{x_i}) + b - 1 \geq 0, \; i = 1, \ldots, m$$

Step 4 *Solving the SVM optimization using Lagrange multipliers, duality principal and KTT conditions*

As proved in [<**empty citation**>], the above optimazation problem is now became **Wolfe dual problem**:

$$\max_{\alpha} \; \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$
$$\text{s.t. } \alpha_i \geq 0, \; \forall i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

where α_i is the Lagrange multiplier of sample x_i.

Step 5 *Using Soft margin SVM.*

The parameter C can be used to let SVM know how much do we let it makes mistakes, but it still need to make as few mistakes as possible.

The Wofle dual problem now become:

$$\max_{\alpha} \; \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$
$$\text{s.t. } \boxed{0 \leq \alpha_i \leq C}, \; \forall i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

The small C will give a wider margin, but with more misclassifications. On the other hand, the big C will make SVM become harder margin classifier, which does not accpect much mistakes.

Step 6 *Using kernel trick*

In case we can not use linear classifier, we need to project the samples to higher dimension and hope that SVM can find a better hyperplane and capable of seperating the classes.

Since we project the samples to other dimension, the Wolfle dual problem now become:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \boxed{K(\boldsymbol{x_i}, \boldsymbol{x_j})}$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

where $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ is $\boldsymbol{x_i} \cdot \boldsymbol{x_j}$ after $\boldsymbol{x_i}, \boldsymbol{x_j}$ have been projected to another dimension. So basically, we only need to find a suitable $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ (or called *kernel*) for an optimal non-linear hyperplane.

Step 7 *Solving Wolfle dual problem*

We can use SMO (sequential minimal optimization) algorithm to solve optimization problem in **step 6**. After finding α, we can compute $\boldsymbol{w}$ and b as following:

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x_i}$$

$$b = \frac{1}{S} \sum_{i=1}^{S} (y_i - w \cdot x_i)$$

where S is the number of support vectors. *Support vectors* are samples that have a positive Lagrange multiplier α_i.

Step 8 *Inference*

The hypothesis function for computing class of a new sample is (using only S support vectors):

$$h(\boldsymbol{x_i}) = \text{sign} \left(\sum_{j=1}^{S} \alpha_j y_j (\boldsymbol{x_j} \cdot \boldsymbol{x_i}) + b \right)$$

Some prominent SVM kernel

Linear kernel

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

This kernel is often used in text classification tasks.

Polynomial kernel

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i} \cdot \boldsymbol{x_j} + c)^d$$

This kernel if often used in case linear seperation is impossible.

Gaussian kernel

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\gamma \parallel \boldsymbol{x_i} - \boldsymbol{x_j} \parallel^2)$$

This kernel project samples to space $\mathcal{R}^\infty$. It use the functions whose value depends only on the distance from some point.

OvR (OvA) and OvO

In case of multiclass classification tasks, we work with more than two sample classes. Because SVM is strict binary classifier, when we need to distinguish n different classes, we have two strategy to use SVM in this task:

One-versus-the-rest (OvR) We train n classifier, the i-th classifier detect if a sample belong to the i class or not. To classify a new sample, put it to n trained classifier and choose the class whose classifier output has the highest score.

One-versus-one (OvO) For each pair (i, j) class in n classes, we train a classifier to detect a sample belong to class i or j. We need to train $n \times (n-1)/2$ classifiers for every pair of class. To classify a new sample, we put it in all trained classifier and see which class "wins the most duels".

SVM is more suitable to OvO strategy since it scales poorly with the size of the training set. Using OvO, a SVM classifier only need to train on a smaller subset of whole dataset.

A-4. Artificial neural network (ANN)

The solution is based on [4].

The original ANN use four seperate classifier for four class respectively. Since 2nd classifier only has 1 training sample, we need to restructure this model such that each classifier can have acceptable number of training samples. We do this by changing this "fat"ANN to "deep"ANN, and creating some new type of classifiers.

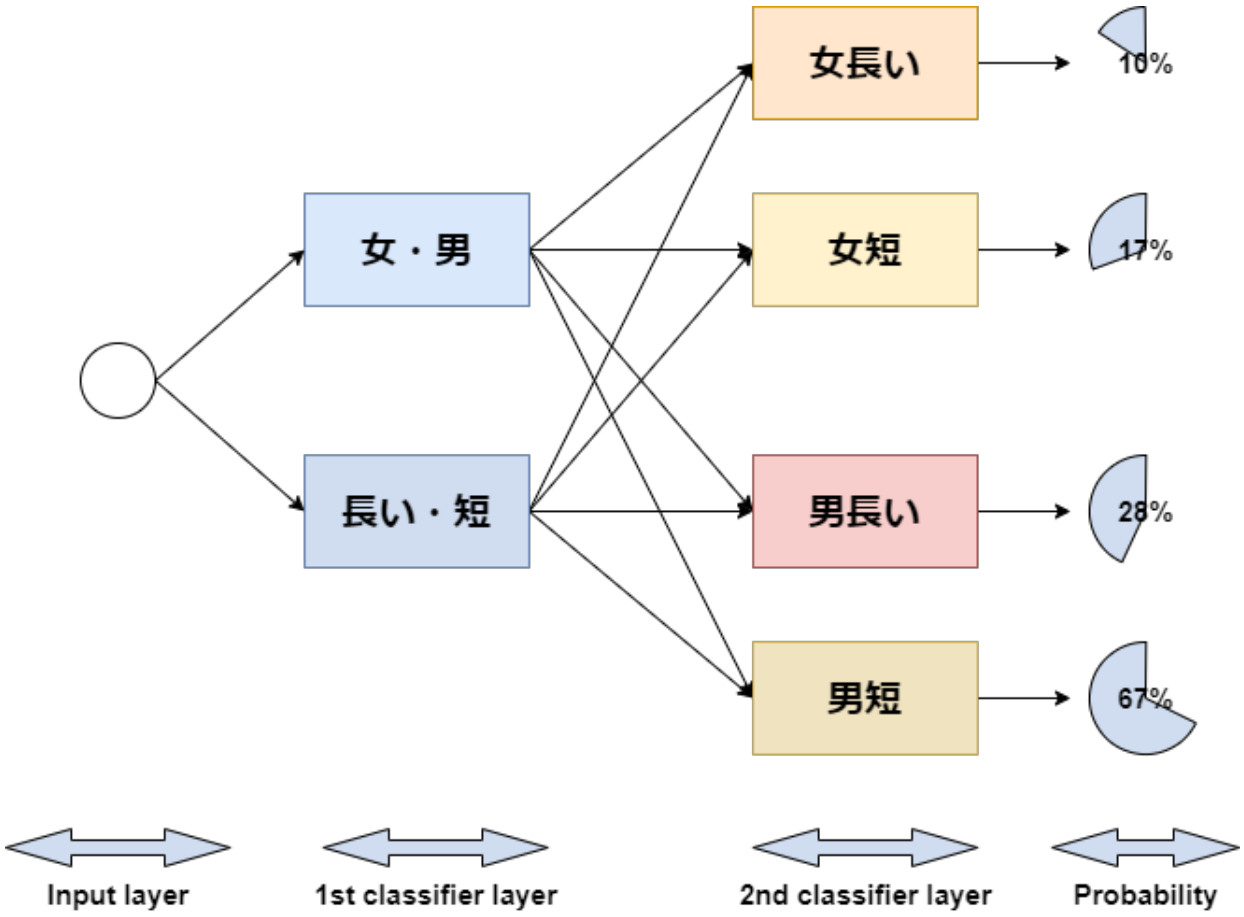
The new model is shown as Image 1. Since we have enormous number of samples of girl/boy class and long/short hair class, first we train two seperate classifiers for these two pair class. Now we do transfer learning by connecting trained 1st classifier layer with 2nd classifier layer. Since the 1st layer has knowlegde about girl/boy pair class and long/short pair class, it can transfer that information directly to 2nd layer. The 2nd layer now only need to find a way to connect results from two classifier in 1st layer and return a probability a sample belong to a class.

The "deep"ANN can use the knowledge from lower layers and build up more complicated structure in higher layers. We can also utilise the pretrained module/layer and corporate them to our new model so we can overcome the case that some class has less number of samples.

A-5. Deep Neural Network (DNN)

a. Unveil the power of deepness

The basic observation suggests that the error rate of testing dataset (or unseen samples) with deep neural network is much smaller than the one of the "fat"neural network, although they may have the same neurons. So to avoid overfitting, we often implement neural network architecture as deep layers.



Hình 1: Architecture of improving classifiers model

However, the most important reason that a DNN can outperform a "fat"ANN is the fact that most of object, data in real world has complex hierachy. The deep nets have the ability to build up these complex hierachy of concepts.

A "fat"ANN need to answer the complicated question at once. Although basically that can be done as proved by Universality theorem, the performance or learning speed is really bad. On the other hand, a DNN can decompose any complicated problems into smaller ones, which are often easier, until these questions are so trivial that DNN can answer with just a single neuron. This is the same as the way human (brain) solve any problem in real life: divide and conquer. This can be seen clearly in the case of visual recognition tasks.

Suppose we need to build a ANN that answer if a given image contains a human face. To make problem easier, we can build the sub-network that answer basic questions. And of course, these questions can even be divided smaller until the neurons work with individual pixels. In other word, the first layer may only need to learn if some pixels make up a basic texture (like straight, corner, etc.). It then transfers these information to higher layer to merge information and answer more detailed question related to shape, color, light, etc.

In short, DNN can capture any hierachy concepts in real life, and build up information more and more complex along the deepness of the network. That is the main reason that DNN can outperform easily a "fat"ANN.

b. Some activation functions

This is shown as the Table 3.

Name	Formula	Advantage	Disadvantages
$Sigmod(z)$	$\frac{1}{1+e^{-z}}$	• Has a well-defined nonzero derivative everywhere. • It also implements same as biological neurons.	• Can cause vanishing gradient problem. • Not zero-centered. • Computationally inefficient when compared to other activation functions.
$Tanh(z)$	$2\sigma(2z) - 1$	• Same as Sigmoid, aslo • Centered around 0, helps speed up convergence.	• Same as Sigmoid, aslo • Centered around 0, helps speed up convergence.
$ReLU(z)$	$\max(0, z)$	• Fast to compute. • Does not have maximum output value helps reduce vanishing gradient.	• Can make gradient bound around 0. • Saturated or "dead"unit.

Bảng 3: Basic specifications of some common activation functions

c. Some optimizers

This is shown as the Table 4.

d. Dropout [2, p. 394 - 399]

Dropout is one of the most powerful regularization techniques to prevent overfitting in DNN. At every training step, every neuron (except output neurons) has a p% chance of being turned off (means they just output 0). They may active again in the following training steps. We call p as *dropout rate*, and it is often set in the range 10%-50%.

The reason droupout makes sense is that when DNN does not work at maximum "performance"(some neurons are turned off), each individual neuron have to be as useful as possible on its own, since now it may not work with its neighboring neurons, and it has to pay utter most attention to each its input neurons. As a result, slight changes in input affect less to the DNN, and the network generalizes better.

At the same time, if we have n neurons that can be turned off, we will have 2^n possible networks, depend which neurons are turned on/off. After training process, we receive a huge number of different networks, but they are not completely independent (since they still share some weights) so the final DNN can be seen as an averaging ensemble of trained droput networks.

Name	Description	Advantage	Disadvantages
Momentum	Care about what the earlier gradient were. It uses gradient as an acceleration, not as a speed. We compute *momentum vector* $\boldsymbol{m} \leftarrow \beta\boldsymbol{m} - \eta\nabla_\theta J(\boldsymbol{\theta})$, where β is *momentum* and update weights $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m}$	Escape from plateus much faster than gradient descent and roll past local optima.	Add another hyperparameter to tune.
RMSProp	Point toward the global optimum from the start. The gradient vector is scaled down by a factor (the learning rate also decays), and it does so faster for steeper dimensions. As a result, we go more directly toward the global optimum. But to avoid going down too quickly and stop (due to loss of learning rate), RMSProp accumulates only recent gradients.	Go more directly to global optimum. It also requires much less tuning of learning rate hyperparameter.	Still slower than more advanced optimizers.
Adam	It keeps track of an exponentially decaying average of past gradient like momentum and keeps track of an exponentially decaying average past squared gradients.	Require less tuning of learning rate.	May lead to solution that generalize poorly on some datasets.

Bảng 4: Basic specifications of some optimizers

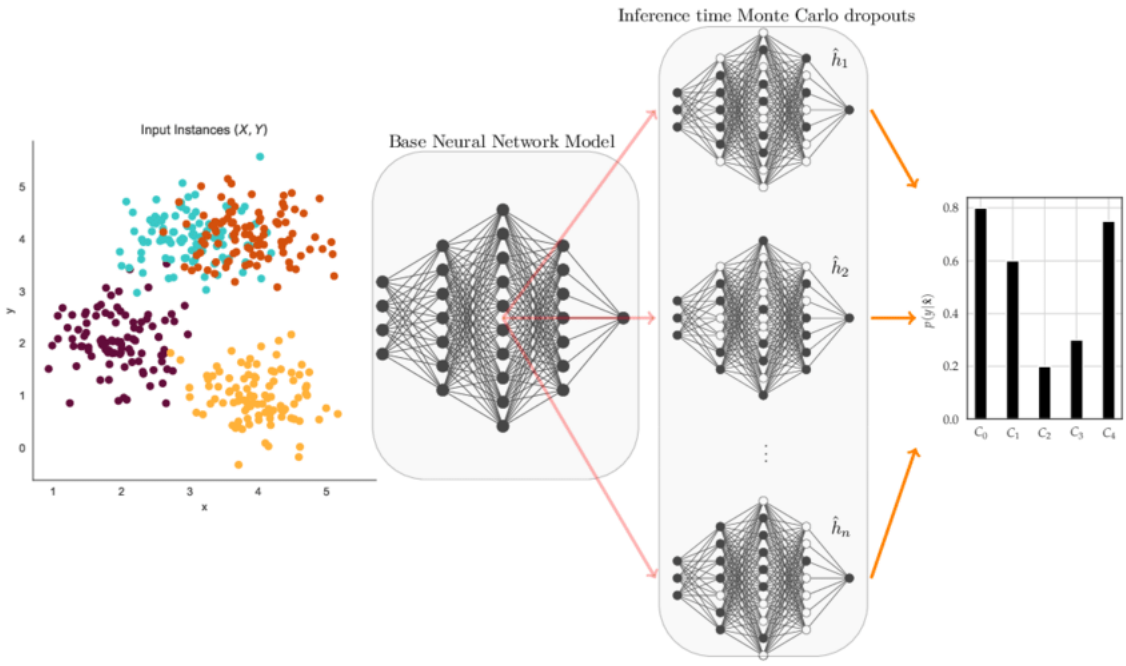
Monte Carlo Dropout

This technique can also boost the performance of any trained dropout DNN without retraining. But moreover, we can use it in the testing/inferencing process and get a better measure of model's uncertainty. It means the model does not fall into the case "to be really confident but wrong". It is utmost important in risk-sensitive or quick-response system. The probability of classification will be not overconfident.

We do this simply by turning on the droup out even during testing process, but we need to predict each class a big number of times and compute avarage result. We need

to balance between the inference time and the accuracy in this case.

Take example in Picture 2. We use the base DNN (which is dropout trained) and infer from the datasets three times, each time with a different sub-model due to dropout. After that we compute avarage classification result of each sample



Hình 2: Monte Carlo Dropout demonstration

B-1. Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models

This section is based mainly on [8]. All formula, information, specifications, etc. are collected from those references. Any other references beside those will be cited in-text directly.

Overview

This model is proposed by Ph.D. Douglas A. Reynolds (a senior member of technical staff at MIT Lincoln Laboratory, also a memmber of IEEE) and Richard C. Rose. It was first published in IEEE Transactions on Speech and Audio processsing (Vol. 3, No. 1), in January 1995. This was one of most famous unsupervised machine learning model used in speaker recognition, and are still used even to this day.

Some applications

1. *Text-Independent speaker identification.*

   GMM can capture the underlying distribution of spectral in voice of each person, which is charactersized by his/her vocal tract shape. As a result, GMM does not need information of sematic to distiguish person's voice. Each speaker registered to the databases and supplied their voice sample will have their own classifier as GMM.

2. *3D face recognition.*

   Using only a single sensor, region covariance matrixes derives low-dimensional feature vectors, which finally modeled by GMM. They are finally then fed to a SVM to retrieve the identity.

3. *Classify the pixel distribution of main brain tissues.*

   Here GMM is used to study the important variability in the mixing probability associated with white matter and grey matter between the left and right hemispheres.

Gaussian Mixture Model

Following are the main steps to train a common GMM (in this problem):

Step 1 *Define the GMM*

Assume that a feature vector respects to an arbitary voice is generated through the following probabilistic process:

$$p(\vec{x}|\lambda) = \sum_{i=1}^{K} p_i b_i(\vec{x})$$

where $\vec{x}$ is a m-dimensional vector (or a feature vector), $b_i(\vec{x})$ is a component density, p_i is the respective mixture weight. Each component densitie $b_i$have a mean $\vec{\mu_i}$ and

a covariance matrix $\mathbf{\Sigma_i}$. This complete Gaussian mixture density is parameterized by $\lambda = \{p_i, \vec{\mu_i}, \mathbf{\Sigma_i}\}$.

The intuition is that each speark will has their own λ, or a Gaussian mixture.

Step 2 *Retrieve the feature vector*

From each utterance, we use MFCC (Mel Frequency Cepstral Coefficient) to retrieve feature vectors. This vector contains information about speactral (frequency range, time, etc.) of important characteristics produced solely by vocal tract of a speaker.

Step 3 *Choose number of clusters*

Choose a suitable value for K.

Step 4 *Training using Expectation-Maximization (EM) algorithm*

First, EM initializes λ randomly, then it repeats two steps until convergence: (1) *Expectation step*: for each instance, the algorithm estimates the probability that it belongs to each cluster, using current cluster λ. (2) *Maximization step* Each cluster is updated using all feature vectors, with each instance weighted by the estimated probability that it belongs to that cluster. It means λ is affected most by instaces have high chance generated by that cluster.

Step 5 *Speaker identification*

A gropu of S speaker is represented by GMM's $\lambda_1, \lambda_2, \ldots, \lambda_S$. In the inference step, given a utterance of a speaker registered before, we will determine:

$$\hat{S} = \max_{1 \le k \le S} \sum_{t=1}^{T} \log p(\vec{x}_t | \lambda_k)$$

where $\hat{S}$ is index of speaker, $\vec{t_t}$ is feature vector retrieved from speaker utterance. $p(\vec{x}_t|\lambda_k)$ can be calculated easily by using formula in **Step 1** with trained λ.

Evaluation

The evaluation is shown as Table 5.

B-2. Voice Math of Google Assistant and Google Home

This section is based mainly on [3]. All formula, information, specifications, etc. are collected from those references. Any other references beside those will be cited in-text directly.

Overview of the solution

IoT developments help every device has the ability to connect to the Internet, to each other and especially interactive with user. Now people can set up a cluster smart device in their home and control them by various means, one of them is by user's voice. However, a family or a place may have lots of people attdending at the same time so it may

Property	Advantages	Disadvantages
Accuracy	High in case normalization, the data has some form of Gaussian distribution. Good for even huge population size	Easy to be affected by outliners, noisy, environmental context. Cannot cope to nonellipsoidal clusters or arbitary shapes.
Running time	Enormous quick.	
Memory usage	Low.	
Library support	Various. Out-of-the-box usage and easy tuning.	
Architecture	Simple, easy to visualize and interpret.	Need to determine suitable number of cluster.

Bảng 5: Evaluate GMM in this speaker recognition system

cause chaos or wrong results if device can not recognize the identity of a voice. Besides that, there is high demanding of passwordless security for covenient and more safety, so voice also become a prominent mean to authenticate the user in a system without using password.

As a result, Google Inc. has integrated a feature called "Voice Math"to their Google Home eco-system and Google Assitant on various device to cope with identification through voice. Every one who is using Android with Google Assistant, Google Home may access to this feature and trained the device to recognize their voice. This wil give various benefits to the user.

Main functionality

Personal results The Google Assistant may show personal results from Gmail, Calendar, Photos, etc; control device in house, car, etc. if it recognizes a registerd voice. In case it receives a strange voice, it will not respond at all.

Payment If we pay through Google Assistant, we can verify the purchases with Voice Math as a kind of authentication.

Evaluation

If it is still developed, this feature can bring more convenient for user in many aspects of living and working. But Google still need to reduce the False Accepting Rate to really bring the true privacy and security to real application.

Advatages	Disadvantages
• High accuracy with popular languages. • Low latency, quick processing. • Just store enough information of user model on local device. • High applicable.	• Inconsistent accuracy with other languages. • Not to be really robust in every context, surrounding environment. • Still can be bypassed by Generative model. • High bias to gender.

Bảng 6: Evaluation Voice Match

Competitor

There are at least two other organization has made the same functionality and incorporate into their main products.

Siri The Assistant of Iphone eco-system.

Cortana Not really robust like above product, it is integrated to the product of Window OS.

Tài liệu

[1] Kowalczyk Alexandre. *Support Vector Machine Succinctly*. Syncfusion, 2017.

[2] Géron Aurélien. *Hands-on Machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems*. English. 2019. ISBN: 9781492032649 1492032646.

[3] Google. *Teach Google Assistant to recognize your voice with Voice Match*. URL: https://support.google.com/assistant/answer/9071681?hl=en%7B%5C&%7Dref%7B%5C_%7Dtopic=7658509%7B%5C&%7Dsjid=17765580829262579022-AP.

[4] Hoang Thai Le. *Bài 8: Mạng nơ-ron nhân tạo dựa theo cách tiếp cận học sâu - Trí tuệ nhân tạo - Nhận dạng (Phần 2)*. 2021. URL: https://www.youtube.com/watch?v=uwvJ8OS-m1Y%7B%5C&%7Dlist=PLFpf6IAKcCjVwoHYSGENBdNLkQVvubEsX%7B%5C&%7Dindex=9%7B%5C&%7Dt=3603s%7B%5C&%7Dab%7B%5C_%7Dchannel=ThaiLeHoang.

[5] A.M. Martinez **and** A.C. Kak. "PCA versus LDA". **in***IEEE Transactions on Pattern Analysis and Machine Intelligence*: 23.2 (2001), **pages** 228–233. ISSN: 01628828. DOI: 10.1109/34.908974. URL: http://ieeexplore.ieee.org/document/908974/.

[6] Matti Pietikäinen **andothers**. "Background". **in**2011: **pages** 3–12. DOI: 10.1007/978-0-85729-748-8_1. URL: http://link.springer.com/10.1007/978-0-85729-748-8%7B%5C_%7D1.

[7] Matti Pietikäinen **andothers**. "Local Binary Patterns for Still Images". **in**2011: **pages** 13–47. DOI: 10.1007/978-0-85729-748-8_2. URL: http://link.springer.com/10.1007/978-0-85729-748-8%7B%5C_%7D2.

[8] D.A. Reynolds **and** R.C. Rose. "Robust text-independent speaker identification using Gaussian mixture speaker models". **in***IEEE Transactions on Speech and Audio Processing*: 3.1 (1995), **pages** 72–83. ISSN: 10636676. DOI: 10.1109/89.365379. URL: http://ieeexplore.ieee.org/document/365379/.

[9] Jonathon Shlens. *A Tutorial on Principal Component Analysis*. 2014. arXiv: 1404.1100 [cs.LG].