# 机器学习导论
# 习题四

141210016, 刘冰楠, bingnliu@outlook.com

2017 年 5 月 17 日

## 1 [20pts] Reading Materials on CNN

卷积神经网络 (Convolution Neural Network, 简称 CNN) 是一类具有特殊结构的神经网络，在深度学习的发展中具有里程碑式的意义。其中，Hinton 于 2012 年提出的AlexNet可以说是深度神经网络在计算机视觉问题上一次重大的突破。

关于 AlexNet 的具体技术细节总结在经典文章“ImageNet Classification with Deep Convolutional Neural Networks”, by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton in NIPS'12，目前已逾万次引用。在这篇文章中，它提出使用 ReLU 作为激活函数，并创新性地使用 GPU 对运算进行加速。请仔细阅读该论文，并回答下列问题 (请用 1-2 句话简要回答每个小问题，中英文均可)。

(a) **[5pts]** Describe your understanding of how ReLU helps its success? And, how do the GPUs help out?

(b) **[5pts]** Using the average of predictions from several networks help reduce the error rates. Why?

(c) **[5pts]** Where is the dropout technique applied? How does it help? And what is the cost of using dropout?

(d) **[5pts]** How many parameters are there in AlexNet? Why the dataset size(1.2 million) is important for the success of AlexNet?

关于 CNN，推荐阅读一份非常优秀的学习材料，由南京大学计算机系吴建鑫教授[1]所编写的讲义 Introduction to Convolutional Neural Networks[2]，本题目为此讲义的 Exercise-5，已获得吴建鑫老师授权使用。

**Solution.**

[1] 吴建鑫教授主页链接为cs.nju.edu.cn/wujx

[2] 由此链接可访问讲义https://cs.nju.edu.cn/wujx/paper/CNN.pdf

## 1.1 Problem (a)

### 1.1.1 Definition of ReLU

ReLu means Rectified Linear Unit. It is a non-saturating function:

$$\lim_{x \to \infty} f(x) = +\infty \tag{1.1}$$

### 1.1.2 How ReLU helps: learn faster

In terms of training time with gradient descent, It learns several times **faster** compared to other saturating activation functions like sigmoid. Fast learning has a great influence on the performance of large models trained on **large datasets** (because we can achieve better performance under the same time)

Learning with ReLU is faster because its derivative is simple enough:

$$\begin{cases} f'(x) = 1, & x > 0 \\ f'(x) = 0, & x \leq 0 \end{cases} \tag{1.2}$$

### 1.1.3 How ReLU helps: avoid gradient vanishing

Another advantage of ReLU is it can **avoid gradient vanishing**. Its gradient is a constant while $f_{sig}(x) \leq 1/4$. Since AlexNet is a **deep** neural network, this feature important.

### 1.1.4 How GPU helps: accelerate the computations

GPU has:

1. high bandwidth main memory;
2. hiding memory access latency under thread parallelism;
3. large and fast register and L1 memory.

Current GPUs are powerful enough to facilitate the training of interestingly-large CNNs. The author's group wrote a highly-optimized GPU **implementation** of 2D convolution and all the other operations inherent in training convolutional neural networks. Therefore they can **learn larger datasets** to train more powerful models and avoid overfitting.

### 1.1.5 How GPU helps: cross-GPU parallelization

Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another' s memory **directly**, without going through host machine memory. Therefore multiple GPUs can be utilized in a single task to further increase computation capability.

### 1.1.6 How GPU helps: GPU-CPU parallelization

In the author's implementation, there are some operations (like label-preserving transformations) executed in Python code on the CPU while the GPU is working on the main job—training. So these those operations are, in effect, computationally free.

## 1.2 Problem (b)

There are two averaging operations in the paper:

1. Averaging on predictions from different inputs;
2. Averaging on predictions from different network models.

The first one works because the input dimension is actually $224x224$ after **data augmentation**. So *At test time, the network makes a prediction by extracting five* $224x224$ *patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.* This helps reduce error rates because with these ten patches of one image, we have a integrated input of it (no loss of information). Also this produces more robust predictions.

The second, averaging of predictions from several networks help reduce error rates because:

**Reduce variance of a specific model (more robust)** A specific trained neural network generally has good expectation performance on training dataset. However, due to variance of the model, it might make bad predictions on some samples. According to formula

$$\mathrm{Var}(\frac{X+Y}{2}) = \frac{1}{4}\mathrm{Var}(X) + \frac{1}{4}\mathrm{Var}(Y) + \frac{2}{4}\mathrm{Cov}(X,Y), \tag{1.3}$$

a combination prediction can have smaller variance.

**Reduce of a specific model (more precise)** Every model has its bias, taking average of them can reduce such biases.

**Avoid overfitting** Different models have different views of the same problem. If we take average of them, it is more likely to get the whole view. Therefore, it is less likely to fall into some features / patterns that only belong to training data, thus avoiding overfitting. Reducing overfitting means we can have better performance on test datasets.

## 1.3 Problem (c)

### 1.3.1 Definition and mechanism

Dropout is a technique for improving neural networks by **reducing overfitting**. Typically, we **set the output of each hidden neuron to zero with a certain probability** when training. The neurons which are "dropped out" in this way do not contribute

to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weight.
At test time, we use **all** the neurons but multiply their outputs by a certain factor, which is related to the probability.

#### 1.3.2 Where it is applied

The author uses dropout in the **first two fully-connected layers** in their CNN architecture.

#### 1.3.3 How does it help

Standard back-propagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout **breaks up these co-adaptations by the presence of any particular hidden unit unreliabl**e. (a neuron cannot rely on the presence of particular other neurons)

#### 1.3.4 Cost of dropout

"Drop out" creates a **trade-off** between overfitting and training time. When it is applied, parameter updates become very noisy. Each training case effectively tries to train a different random architecture. Therefore, the gradients that are being computed are not gradients of the final architecture that will be used at test time. Therefore the training time becomes (typically 2-3 times) longer.

In the paper, the author uses a **probability of 0.5** to set output zero. Thus dropout **roughly doubles the number of iterations** required to converge.

However, as said by the author, this cost is still less expensive than training many different models and take average.

### 1.4 Problem (d)

#### 1.4.1 Number of parameters

The neural network has **60 million parameters** and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax.

#### 1.4.2 Why dataset size is important

Three ways to improve performance:

1. collect larger datasets;
2. learn more powerful models;

3. use better techniques for preventing overfitting.

In the context of image processing (classification), objects in realistic settings exhibit considerable **variability**, so to learn to recognize them it is necessary to use **a model with strong enough learning capability**. We can see large datasets size has two importance:

- A model has to be trained on large enough datasets to **recognize things with so much diversity and variability**;
- A model has to be trained on large enough datasets to **learn so many parameters without considerable overfitting**.

The shortcoming of small datasets has been widely recognized (Pinto et al. (2008)[2]). For example, objects in small datasets do not have enough variations in object view, position, size, etc.

# 2 [20pts] Kernel Functions

(1) 试通过定义证明以下函数都是一个合法的核函数：

(i) **[5pts]** 多项式核: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^d$;

(ii) **[10pts]** 高斯核：$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$, 其中 $\sigma > 0$.

(2) **[5pts]** 试证明 $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{1+e^{-\mathbf{x}_i^T\mathbf{x}_j}}$ 不是合法的核函数。

## 2.1 Problem 2.1 Polynomial Kernel

**Proof.**

### 2.1.1 Expansion of Polynomial Kernel

**Multinomial theorem** states that

$$(x_1 + x_2 + \cdots + x_m)^n = \sum_{k_1+k_2+\cdots+k_m=n} \binom{n}{k_1, k_2, \ldots, k_m} \prod_{t=1}^{m} x_t^{k_t}. \tag{2.1}$$

Therefore we can expand the polynomial kernel function as:

$$\begin{aligned}
\kappa(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^d \\
&= (\sum_{k=1}^{m} x_{ik}x_{jk})^d \\
&= \sum_{k_1+k_2+\cdots+k_m=d} \binom{d}{k_1, k_2, \ldots, k_m} \prod_{t=1}^{m} (x_{it}x_{jt})^{k_t} \\
&= \sum_{k_1+k_2+\cdots+k_m=d} \binom{d}{k_1, k_2, \ldots, k_m} (\prod_{t=1}^{m} x_{it}^{k_t})(\prod_{t=1}^{m} x_{jt}^{k_t}).
\end{aligned} \tag{2.2}$$

Using **Stars and bars** technique in combinatorics, we can count that there are totally $D = \binom{d+m-1}{m-1}$ terms in expansion(2.2). Consider a map $\Phi$,

$$\begin{aligned} \Phi : \mathbb{R}^m &\to \mathbb{R}^D \\ \mathbf{x} &\to \Phi(\mathbf{x}) \\ \text{s.t. } \Phi(\mathbf{x})_i &= \sqrt{\binom{d}{k_{i_1}, k_{i_2}, \ldots, k_{i_m}}} \prod_{t=i_1}^{i_m} x_t^{k_t}, \end{aligned} \tag{2.3}$$

where the tuple $(k_{i_1}, k_{i_2}, \ldots, k_{i_m})$ is one of all sets of numbers that satisfy $k_1 + k_2 + \cdots + k_m = d$. Then we can verify that

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle = \sum_{k=1}^{D} \Phi(\mathbf{x}_i)_k \Phi(\mathbf{x}_j)_k. \tag{2.4}$$

### 2.1.2 Proof of Positive Semi-definiteness of The Kernel

Let $\mathbf{K}$ be the kernel matrix (gram matrix), then $\forall \mathbf{x}_i, \mathbf{x}_j, \mathbf{a} \in \mathbb{R}^m$:

$$\begin{aligned} \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a} &= \sum_{i=1}^{m}\sum_{j=1}^{m} a_i \kappa(\mathbf{x}_i, \mathbf{x}_j) a_j \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} a_i \sum_{k=1}^{D} \Phi(\mathbf{x}_i)_k \Phi(\mathbf{x}_j)_k a_j \\ &= \sum_{k=1}^{D}\sum_{i=1}^{m}\sum_{j=1}^{m} a_i \Phi(\mathbf{x}_i)_k \Phi(\mathbf{x}_j)_k a_j \\ &= \sum_{k=1}^{D}(\sum_{i=1}^{m} a_i \Phi(\mathbf{x}_i)_k)(\sum_{j=1}^{m} a_j \Phi(\mathbf{x}_j)_k) \\ &= \sum_{k=1}^{D}(\sum_{i=1}^{m} a_i \Phi(\mathbf{x}_i)_k)^2 \\ &= \left\| \sum_{i=1}^{m} a_i \Phi(\mathbf{x}_i) \right\|^2 \\ &\geq 0. \end{aligned} \tag{2.5}$$

Therefore Polynomial Kernel is positive definite, thus by definition a legal kernel. □

## 2.2 Problem 2.2 Gaussian Kernel

**Proof.**

### 2.2.1 Expansion of Euclidean Distance

We have

$$\begin{aligned}\|\mathbf{x}_i - \mathbf{x}_j\|^2 &= (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j) \\ &= \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i + \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_j - 2\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j \\ &= \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j.\end{aligned} \tag{2.6}$$

### 2.2.2 Proof of Positive Semi-definiteness of The Kernel

Let $\gamma = 1/2\sigma^2$ for convenience. Then $\forall \mathbf{x}_i, \mathbf{x}_j, \mathbf{a} \in \mathcal{R}^m$:

$$\begin{aligned}\mathbf{a}^{\mathrm{T}}\kappa(\mathbf{x}_i, \mathbf{x}_j)\mathbf{a} &= \sum_{i=1}^{m}\sum_{j=1}^{m} a_i \kappa(\mathbf{x}_i, \mathbf{x}_j) a_j \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} a_i \exp(\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2) a_j \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} a_i [\exp(-\gamma\|\mathbf{x}_i\|^2) * \exp(-\gamma\|\mathbf{x}_i\|^2) * \exp(2\gamma\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)] a_j \\ &= \sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) * \exp(2\gamma\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j) \\ &= \sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) * \sum_{q=0}^{\infty}(2\gamma\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^q \\ &= \sum_{q=0}^{\infty}(2\gamma)^q \sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) * (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^q.\end{aligned} \tag{2.7}$$

According to conclusion of previous section (See Eq.(2.3)), we can represent $(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^q$ as:

$$(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^q = \langle \Phi^q(\mathbf{x}_i), \Phi^q(\mathbf{x}_j) \rangle = \sum_{k=1}^{D} \Phi^q(\mathbf{x}_i)_k \Phi^q(\mathbf{x}_j)_k. \tag{2.8}$$

Note in Eq.(2.8), whether the power is $d$ or $q$ does not make substantial difference. Therefore we have:

$$\begin{aligned}
\mathbf{a}^{\mathrm{T}}\kappa(\mathbf{x}_i,\mathbf{x}_j)\mathbf{a} &= \sum_{q=0}^{\infty}(2\gamma)^q \sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) * (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^q \\
&= \sum_{q=0}^{\infty}(2\gamma)^q \sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{k=1}^{D} \Phi^q(\mathbf{x}_i)_k \Phi^q(\mathbf{x}_j)_k \\
&= \sum_{q=0}^{\infty}(2\gamma)^q \sum_{k=1}^{D}\sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) \Phi^q(\mathbf{x}_i)_k \Phi^q(\mathbf{x}_j)_k \\
&= \sum_{q=0}^{\infty}(2\gamma)^q \sum_{k=1}^{D} \left(\sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \Phi^q(\mathbf{x}_i)_k\right) \left(\sum_{j=1}^{m} a_j \exp(-\gamma\|\mathbf{x}_i\|^2) \Phi^q(\mathbf{x}_j)_k\right) \\
&= \sum_{q=0}^{\infty}(2\gamma)^q \sum_{k=1}^{D} \left(\sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \Phi^q(\mathbf{x}_i)_k\right)^2 \\
&= \sum_{q=0}^{\infty}(2\gamma)^q \left\|\sum_{i=1}^{m} a_i \exp(-\gamma\|\mathbf{x}_i\|^2) \Phi^q(\mathbf{x}_i)\right\|^2 \\
&\geq 0.
\end{aligned} \tag{2.9}$$

Therefore Gaussian Kernel is positive definite, thus by definition a legal kernel. □

### 2.3 Problem 2.3 An Illegal Kernel

**Proof.** We need to point out only one counter example.

We have two samples: $\mathbf{x}_1 = [1,1]^{\mathrm{T}}, \mathbf{x}_2 = [2,2]^{\mathrm{T}}$. Then the gram matrix is:

$$\begin{bmatrix} \kappa(\mathbf{x}_1,\mathbf{x}_1) & \kappa(\mathbf{x}_1,\mathbf{x}_2) \\ \kappa(\mathbf{x}_2,\mathbf{x}_1) & \kappa(\mathbf{x}_2,\mathbf{x}_2) \end{bmatrix} \tag{2.10}$$

numerical result is:

$$\begin{bmatrix} 0.8808 & 0.9820 \\ 0.9820 & 0.9997 \end{bmatrix} \tag{2.11}$$

whose determinant is $-0.838 < 0$. A positive definite matrix holds the property that **its leading principal minors are all positive**. So Matrix.(2.10) can't be positive definite.

Therefore This kernel is not positive definite, thus by definition an illegal kernel. □

## 3 [25pts] SVM with Weighted Penalty

考虑标准的 SVM 优化问题如下 (即课本公式 (6.35))，

$$\begin{aligned}
\min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \\
\text{s.t.} \quad & y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 - \xi_i \\
& \xi_i \geq 0, i = 1, 2, \cdots, m.
\end{aligned} \tag{3.1}$$

注意到，在(3.1)中，对于正例和负例，其在目标函数中分类错误的“惩罚”是相同的。在实际场景中，很多时候正例和负例错分的“惩罚”代价是不同的，比如考虑癌症诊断，将一个确实患有癌症的人误分类为健康人，以及将健康人误分类为患有癌症，产生的错误影响以及代价不应该认为是等同的。

现在，我们希望对负例分类错误的样本 (即 false positive) 施加 $k > 0$ 倍于正例中被分错的样本的“惩罚”。对于此类场景下，

(1) **[10pts]** 请给出相应的 SVM 优化问题;

(2) **[15pts]** 请给出相应的对偶问题，要求详细的推导步骤，尤其是如 KKT 条件等。

**Solution.**

Let's consider the **generalized** version — to **assign different weights to samples**:

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} C_i \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, 2, \cdots, m. \end{aligned} \tag{3.2}$$

Problem Eq.(3.1) is just a special case of Eq.(3.2) when we only have two different values of weights:

$$\begin{cases} C_i^- = kC, & \text{for true negative samples} \\ C_i^+ = C, & \text{for true positive ones} \end{cases} \tag{3.3}$$

or in a more compact form:

$$C_i = \frac{y_i(1-k) + k + 1}{2} C \tag{3.4}$$

For example, when a true negative sample $\mathbf{x}_i$ will **always** have $C_i = kC$. When it is correctly classified, $\xi_i = 0$ and $C_i\xi_i = 0$; when it is not, $\xi_i > 0$ and it is punished.

Here we will derive the solution **under general circumstances and apply it to this special case**.

## 3.1 Problem 3.1 - SVM Optimization Problem (general case)

i.e. (3.2)

## 3.2 Problem 3.1 - SVM Optimization Problem (special case)

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \frac{y_i(1-k) + k + 1}{2} \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, 2, \cdots, m. \end{aligned} \tag{3.5}$$

## 3.3 Problem 3.2 - Dual And KKT (general case)

### 3.3.1 Dual Problem Derivation

We continue with the general case (3.2). First write the Lagarangian of (3.2):

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} C_i\xi_i + \sum_{i=1}^{m} \alpha_i[1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)] - \sum_{i=1}^{m} \beta_i\xi_i \tag{3.6}$$

The primal problem is:

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \max_{\alpha,\beta} \quad & \mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) \\ \text{s.t.} \quad & \alpha_i \geq 0 \\ & \beta_i \geq 0, i = 1, 2, \cdots, m. \end{aligned} \tag{3.7}$$

The dual problem is:

$$\begin{aligned} \max_{\alpha,\beta} \min_{\mathbf{w},b,\xi_i} \quad & \mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) \\ \text{s.t.} \quad & \alpha_i \geq 0 \\ & \beta_i \geq 0, i = 1, 2, \cdots, m. \end{aligned} \tag{3.8}$$

where the *dual function* is:

$$\Gamma(\alpha, \beta) = \min_{\mathbf{w},b,\xi_i} \quad \mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) \tag{3.9}$$

To calculate the expression of dual function, take derivatives:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} & = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \\ \frac{\partial \mathcal{L}}{\partial b} & = \sum_{i=1}^{m} \alpha_i y_i \\ \frac{\partial \mathcal{L}}{\partial \xi_i} & = C_i - \alpha_i - \beta_i = 0 \end{cases} \tag{3.10}$$

i.e.

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{3.11}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{3.12}$$

$$\alpha_i + \beta_i = C_i \tag{3.13}$$

insert Eq.(3.11), (3.12) and (3.13) into Eq.(3.9), we have the **dual problem**:

$$\begin{aligned} \max_{\alpha,\beta} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C_i \\ & \sum_{i=1}^{m} \alpha_i y_i = 0, \ i = 1, 2, \cdots, m. \end{aligned} \tag{3.14}$$

where we use Eq.(3.13) to get $\alpha_i \leq C_i$

### 3.3.2 KKT Conditions

KKT conditions are not just a bunch of equations. They can be divided into **groups** with meaningful interpretation. For **a general standard optimization problem** to minimize $f$ with equality constraints $h_j$ and inequality constraints $g_i$, KKT conditions are:

**stationarity** derivative of Lagrangian = 0

$$\nabla f(x^*) + \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{\ell} \lambda_j \nabla h_j(x^*) = 0 \tag{3.15}$$

**primal feasibility** constraints of the primal problem satisfied

$$\begin{cases} g_i(x^*) \leq \quad 0, \text{ for } i = 1, \ldots, m \\ h_j(x^*) = \quad 0, \text{ for } j = 1, \ldots, \ell \end{cases} \tag{3.16}$$

**dual feasibility** constraints on KKT multipliers

$$\mu_i \geq 0, \text{ for } i = 1, \ldots, m \tag{3.17}$$

**complementary slackness** from which we have sparse solution

$$\mu_i g_i(x^*) = 0, \text{ for } \ i = 1, \ldots, m. \tag{3.18}$$

For problem (3.14), corresponding KKT conditions are:

**stationarity**

i.e. Eq.(3.11), (3.12) and (3.13)

**primal feasibility**

$$\begin{cases} 1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \leq 0 \\ \xi_i \geq 0, \ i = 1, \ldots, m \end{cases} \tag{3.19}$$

**dual feasibility**

$$\begin{cases} \alpha_i \quad \geq 0 \\ \beta_i \quad \geq 0, \ i = 1, \ldots, m \end{cases} \tag{3.20}$$

or equivalently:

$$0 \leq \alpha_i \leq C_i, \ i = 1, \ldots, m \tag{3.21}$$

**complementary slackness**

$$\begin{cases} \alpha_i[1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)] = 0 \\ \beta_i \xi_i = 0, \ i = 1, \ldots, m \end{cases} \tag{3.22}$$

So KKT conditions for (3.14) can be summarized as:

$$\begin{cases} \sum_{i=1}^{m} \alpha_i y_i = 0 \\ 1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \leq 0 \\ \xi_i \geq 0 \\ 0 \leq \alpha_i \leq C_i \\ \alpha_i[1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)] = 0 \\ \beta_i \xi_i = 0, \ i = 1, \ldots, m \end{cases} \tag{3.23}$$

Eq.(3.11) and (3.13) do not appear because they have been implicitly used.

### 3.4 Problem 3.2 - Dual And KKT (special case)

Just insert Eq.(3.4) into Eq.(3.23), then we have:

$$\begin{cases} \sum_{i=1}^{m} \alpha_i y_i = 0 \\ 1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \leq 0 \\ \xi_i \geq 0 \\ 0 \leq \alpha_i \leq \frac{y_i(1-k)+k+1}{2} C \\ \alpha_i[1 - \xi_i - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)] = 0 \\ \beta_i \xi_i = 0, \ i = 1, \ldots, m \end{cases} \tag{3.24}$$

## 4 [35pts] SVM in Practice - LIBSVM

支持向量机 (Support Vector Machine，简称 SVM) 是在工程和科研都非常常用的分类学习算法。有非常成熟的软件包实现了不同形式 SVM 的高效求解，这里比较著名且常用的如 LIBSVM[3]。

(1) **[20pts]** 调用库进行 SVM 的训练，但是用你自己编写的预测函数作出预测。

(2) **[10pts]** 借助我们提供的可视化代码，简要了解绘图工具的使用，通过可视化增进对 SVM 各项参数的理解。详细编程题指南请参见链接：`http://lamda.nju.edu.cn/ml2017/PS4/ML4_programming.html`.

(3) **[5pts]** 在完成上述实践任务之后，你对 SVM 及核函数技巧有什么新的认识吗？请简要谈谈。

**Solution.**

### 4.1 Problem 4.1 My Own Prediction Function

#### 4.1.1 Code

See `main.py` for code.

[3]LIBSVM 主页课参见链接：`https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

#### 4.1.2 Some Notes Worth Taking Care of

**how to implement prediction function** Check API doc. Get necessary data from that class object, then use the data to do calculations.

**about sklearn** Among data members of `clf`, there are `clf.dual_coefficient_` and `clf._dual_coefficient_`, `clf.indtercept_` and `clf._intercept_`. Each pair of values have opposite sign. These are just implementation hacks. **All we need to know is which is the correct one** (the one corresponding to mathematical models) — those that don't start with underscore.

Another point is that `clf.dual_coefficient_` is not $\alpha_i$, but $alpha_i * y_i$.

**data pre-processing** For binary classification, input data should better be $(0, 1)$, not $(-1, 1)$.

**performance** calculate distance matrix (kernel matrix which elements are pair-wise distances) using mathematical expansion when power=2. otherwise we can only loop in C. Using broadcast or apply: f_arr = dual_coef * kernel_res

### 4.2 Problem 4.2 Visualization

Result: see Fig.(1)
API used:

**marker** use `marker` parameter to set shape of markers.

**marker size** use `s` parameter to set different sizes of markers.

**marker color** use `color` parameter to give different color for S.V. and non-S.V.

**transparency** use `alpha` parameter to set transparency of markers.

**edgecolor** use `edgecolor` parameter to set color of edge of markers.

**common title** use `suptitle` parameter to give a common title to all subplots.

### 4.3 Problem 4.3 New Insights

**稀疏性** 在 `clf.support_vectors_` 仅保存着作为支持向量的样本，然而只使用这些和他们对应的系数即可完成预测，此即 SVM 的稀疏性；

**计算瓶颈** 由于训练中需要计算核矩阵 (即 Gram Matrix)，即计算训练样本两两之间的核函数值，这部分是 SVM 最大的计算瓶颈，而且无法通过改进算法优化，只能考虑并行等；

**RBF 核的 $\gamma$ 参数** RBF 核通过指数衰减来衡量两个样本点之间的相似度，$\gamma$ 是指数项的系数，表征了相似度的衰减快慢，过慢的衰减速度无法很好地区分样本点，过快的衰减速度则倾向于把每个样本点分为一类，从而导致过拟合，参见图 (2)。我们可以根据手头样本间距离的分布选择合适的 $\gamma$ 值；

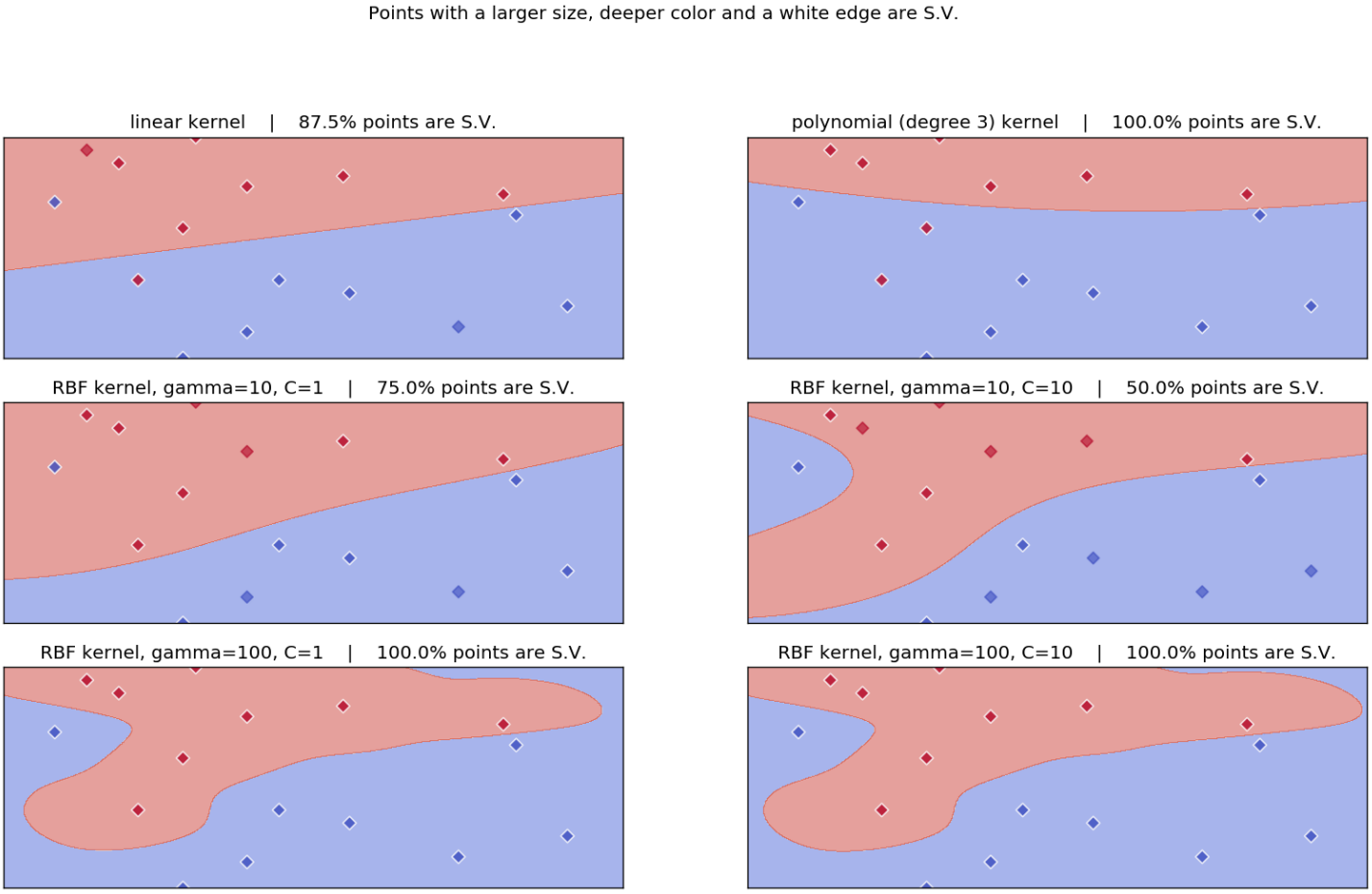


图 1: Visualization of Support Vectors

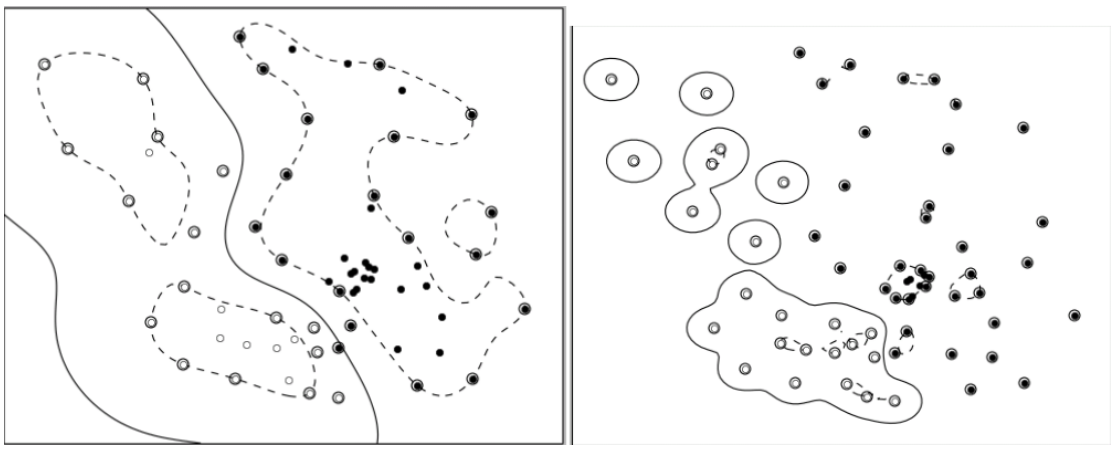

图 2: 右边的 gamma 值大于左边 gamma 值

**软间隔 SVM 的 $C$ 参数** 。从主问题的角度看，$C$ 表征了 bias-variance trade-off，因而较小导致欠拟合，较大导致过拟合；从对偶问题的角度看，$C$ 值是每个样本点的对偶系数的上限，表征了我们能够给予单个样本的 “最大话语权”。

# 参考文献


[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] James J. DiCarlo Nicolas Pinto, David D. Cox. Why is real-world visual object recognition hard? *PLoS Comput Biol*, 4(1):e27, 2008.