Jinchao Xu

Deep Learning Algorithms and Analysis

Summer 2020

Contributors:

This set of notes are based on contributions from many of graduate students, post-doctoral fellows and other collaborators. Here is a partial list:

Juncai He, Qingguo Hong, Li Jiang, Jonathan Siegel, Limin Ma, Lian Zhang, Shenglan Zhao.....

Contents

1

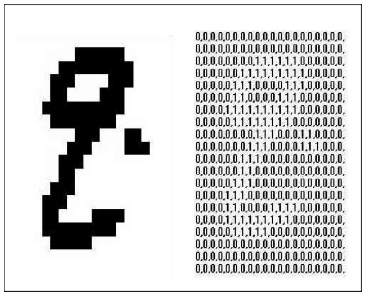
Machine Learning and Image Classification

1.1 A basic machine learning problem: image classification

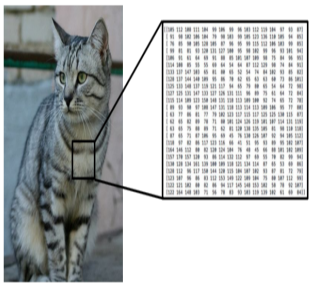
A basic AI problem: classification

- Can a machine (function) tell the difference ?

Mathematically, gray-scale image can be just taken as matrix in $\mathbb{R}^{n_0 \times n_0}$

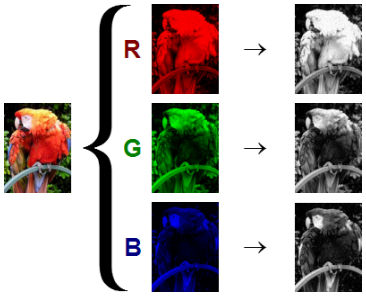


The next figure shows different result from: human vision and computer representation:



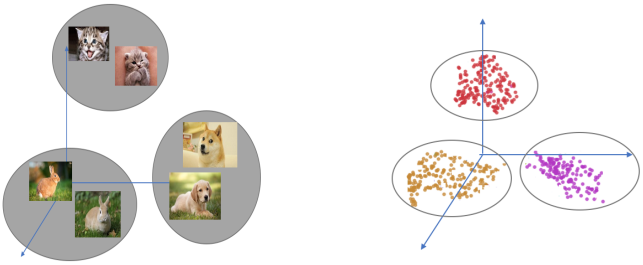
- An image is just a big grid of numbers between [0, 255]
  - e.g. 800×600×3 (3 channels RGB)

Furthermore, color image can be taken as 3D tensor (matrix with 3 channel (RGB)) in $\mathbb{R}^{n_0 \times n_0 \times 3}$



Then, let think about the general supervised learning case.

- Each image = a big vector of pixel values
  - $d = 1280 \times 720 \times 3$ (width× height × RGB channel) ≈ 3M.
- 3 different sets of points in $\mathbb{R}^d$, are they separable?



- Mathematical problem: Find $f(\cdot;\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}^3$ such that:

$$f(\text{  };\boldsymbol{\Theta}) \approx \begin{pmatrix}1\\0\\0\end{pmatrix} \quad f(\text{  };\boldsymbol{\Theta}) \approx \begin{pmatrix}0\\1\\0\end{pmatrix} \quad f(\text{  };\boldsymbol{\Theta}) \approx \begin{pmatrix}0\\0\\1\end{pmatrix}$$

  – Function interpolation
  – data fitting

  Then, the next question is how to formulate "learning"?

- Data: $\{x_j, y_j\}_{j=1}^N$
- Find f^* in some function class such that $f^*(x_j) \approx y_j$.
- Mathematically: solve the optimization problem by parameterizing the abstract function class

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{1.1}$$

  where

$$\mathcal{L}(\boldsymbol{\theta}) := \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(f(x;\boldsymbol{\theta}), y)] \approx L(\boldsymbol{\theta}) := \frac{1}{N}\sum_{j=1}^{N} \ell(y_j, f(x_j;\boldsymbol{\theta}))$$

  Here $\ell(y_j, f(x_j;\boldsymbol{\theta}))$ is the general distance between real label y_j and predicted label $f(x_j;\boldsymbol{\theta})$. Two commonly used distances are
  – ℓ^2 distance:

$$\ell(y_j, f(x_j;\boldsymbol{\theta})) = \|y_j - f(x_j;\boldsymbol{\theta})\|^2.$$

  – KL-divergence distance:

$$\ell(y_j, f(x_j;\boldsymbol{\theta})) = \sum_{i=1}^{k} [y_j]_i \log \frac{[y_j]_i}{[f(x_j;\boldsymbol{\theta})]_i}.$$

- Application: image classification:

$$f(\text{  };\boldsymbol{\theta}) = \begin{pmatrix}0.7\\0.2\\0.1\end{pmatrix} \implies \text{  } = \text{cat}$$

1.2 Image classification problem

We consider a basic machine learning problem for classifying a collection of images into k distinctive classes. As an example, we consider a two-dimensional image which is usually represented by a tensor

$$x \in \mathbb{R}^{n_0 \times n_0 \times c} = \mathbb{R}^d.$$

Here $n_0 \times n_0$ is the original image resolution and

$$c = \begin{cases} 1 \text{ for grayscale image,} \\ 3 \text{ for color image.} \end{cases} \tag{1.2}$$

A typical supervised machine learning problem begins with a data set (training data)

$$D := \{(x_j, y_j)\}_{j=1}^N,$$

with

$$A = \{x_1, x_2, \cdots, x_N\} \quad \text{with} \quad A = A_1 \cup A_2 \cup \cdots \cup A_k,\ A_i \cap A_j = \emptyset, \forall i \neq j.$$

and $y_j \in \mathbb{R}^k$ is the label for data x_j, with $y_i[i]$ as the probability for x_i in classes i or $x_j \in A_i$.

Here for image classification problem,

$$y_j = e_{i_j}, \tag{1.3}$$

if $x_j \in A_{i_j}$ or we say x_j has real label i_j.

Roughly speaking, a supervised learning problem can be thought as a data fitting problem in a high dimensional space $\mathbb{R}^d$. Namely, we need to find a mapping $f : \mathbb{R}^d \mapsto \mathbb{R}^k$, such that, for a given data (x_j, y_j),

$$f(x_j) \approx y_j = e_{i_j} \in \mathbb{R}^k, \tag{1.4}$$

for all $x_j \in A$. For the general setting above, we use a probatilistic model for understanding the output $f(x) \in \mathbb{R}^k$ as a discrete distribution on $\{1, \cdots, k\}$, with $[f(x)]_i$ as the probability for x in the class i, namely

$$0 \leq [f(x)]_i \leq 1, \quad \sum_{i=1}^k [f(x)]_i = 1. \tag{1.5}$$

At last, we finish our model with a simple strategy to choose

$$\arg\max_i \{[f(x)]_i \ : \ i = 1 : k\}, \tag{1.6}$$

as the label for a test data x, which ideally is close to (1.4). The remaining key issue is the construction of the classification mapping f.

Generally speaking, there will be a test set

$$T = \{(x_j, y_j)\}_{j=1}^M, \tag{1.7}$$

with the same dimension of training data D, but is not known before we finish the training process. That is to say, we can use this test data T to verify the performance of trained model f.

1.3 Some popular data sets in image classification

In this subsection, we will introduce some popular and standard data sets in image classification.

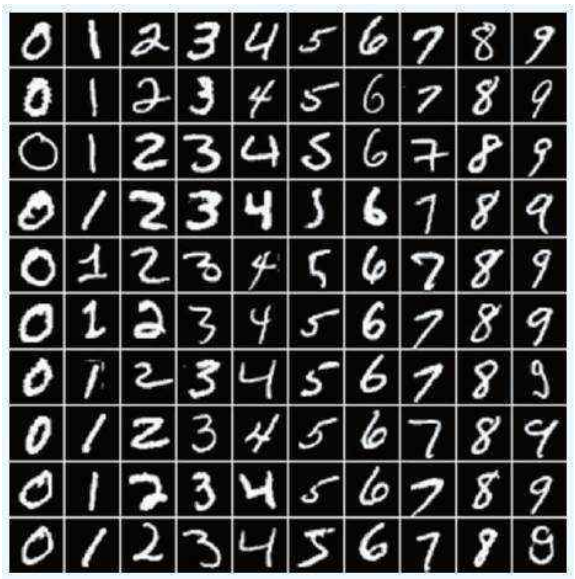
dataset	training (N)	test (M)	classes (k)	channels (c)	input size (d)
MNIST	60K	10K	10	Greyscale	28*28
CIFAR-10	50K	10K	10	RGB	32*32
CIFAR-100	50K	10K	100	RGB	32*32
ImageNet	1.2M	50K	1000	RGB	224*224

Table 1.1. Basic descriptions about popular datasets

1.3.1 MNIST(Modified National Institute of Standards and Technology Database)

This is a database for handwritten digits

- Training set : $N = 60,000$
- Test set : $M = 10,000$
- Image size : $d = 28 * 28 * 1 = 784$
- Classes: $k = 10$



$$x = \text{  } = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{pmatrix} \in \mathbb{R}^{784}$$

(1.8) $$A_1 = \left\{ \text{  }, \cdots \right\} \subset \mathbb{R}^{784}$$

(1.9) $$A_2 = \left\{ \text{  }, \cdots \right\} \subset \mathbb{R}^{784}$$

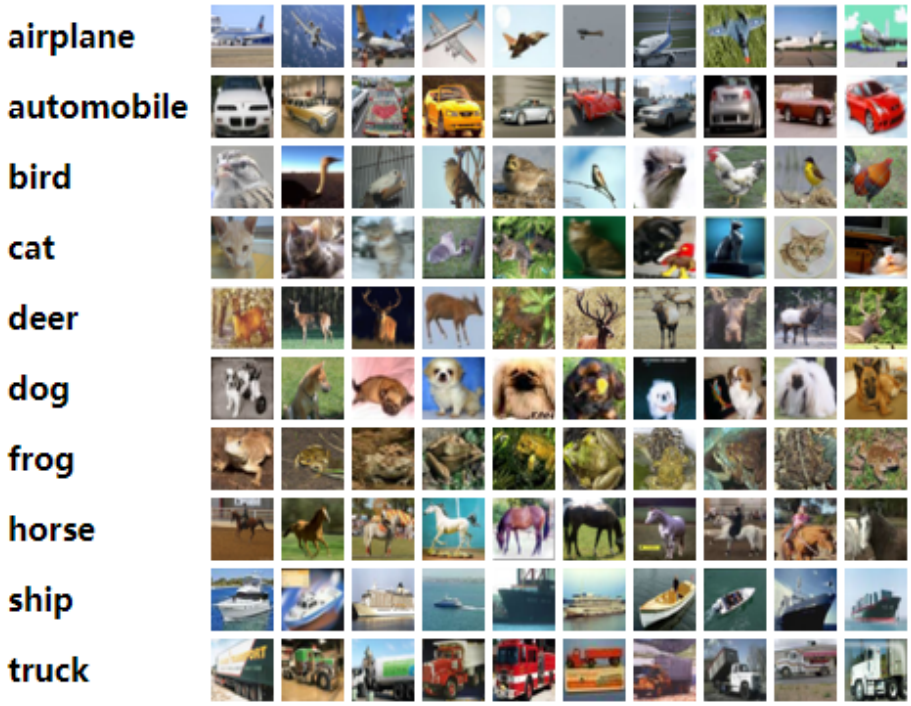
(1.10) $$A_9 = \left\{ \text{  }, \cdots \right\} \subset \mathbb{R}^{784}$$

(1.11) $$A_{10} = \left\{ \text{  }, \cdots \right\} \subset \mathbb{R}^{784}$$

1.3.2 CIFAR

CIFAR-10

- Training set : $N = 50,000$
- Test set : $M = 10,000$
- Image size : $d = 32 * 32 * 3$
- Classes: $k = 10$

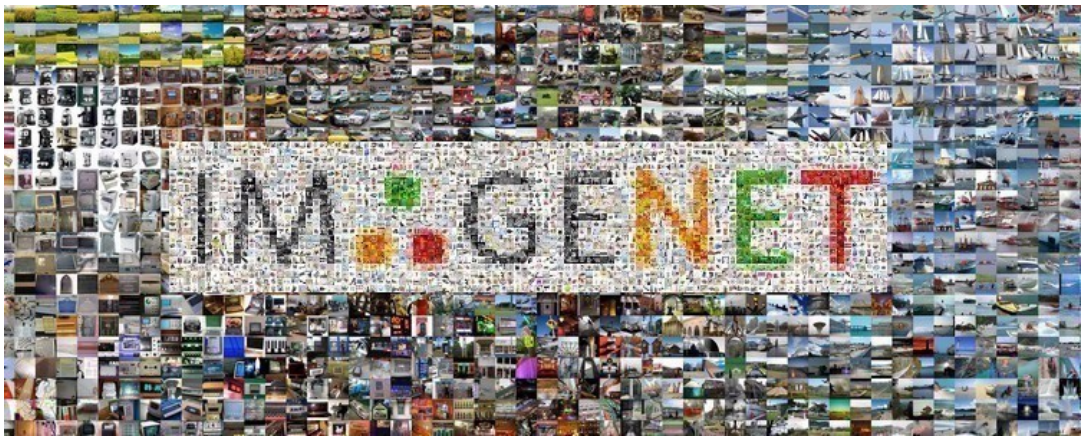


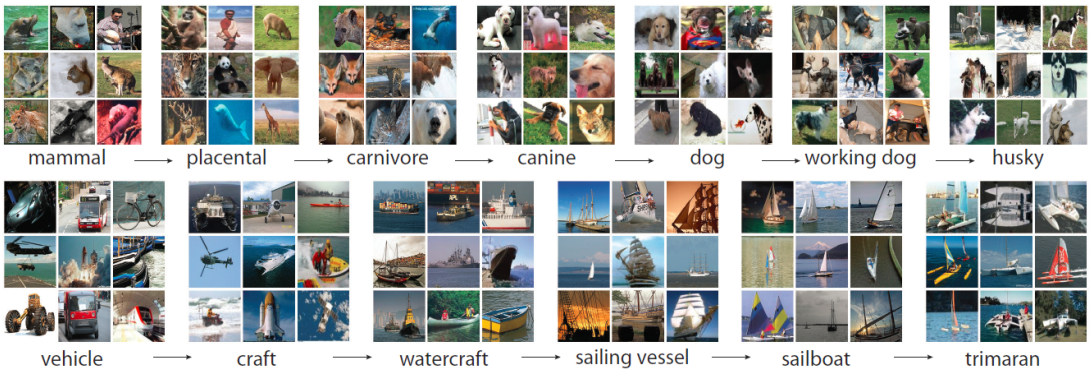
CIFAR-100

- Training set : $N = 50,000$
- Test set : $M = 10,000$
- Image size : $d = 32 * 32 * 3$
- Classes: $k = 100$

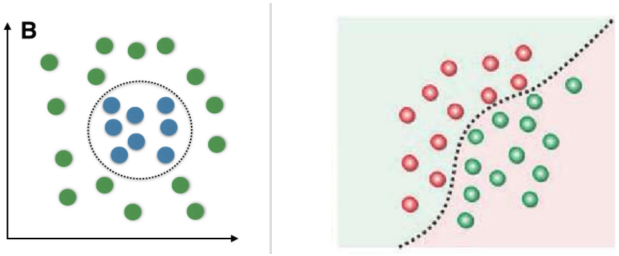
1.3.3 ImageNet

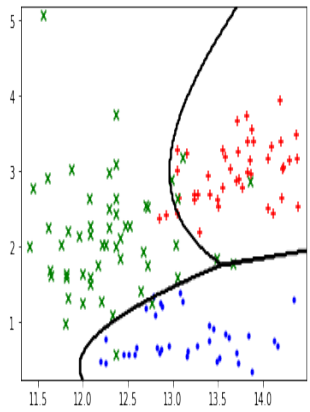
- All data set: $N + M = 1,431,167$
- Image size : $d = 224 * 224 * 3$
- Classes: $k = 1,000$





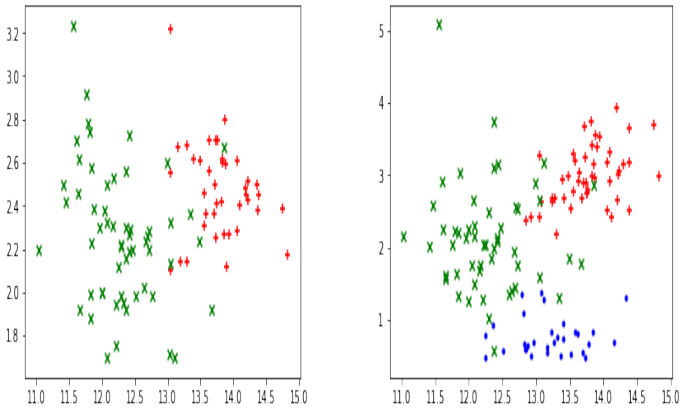
1.4 Classification and decision boundaries



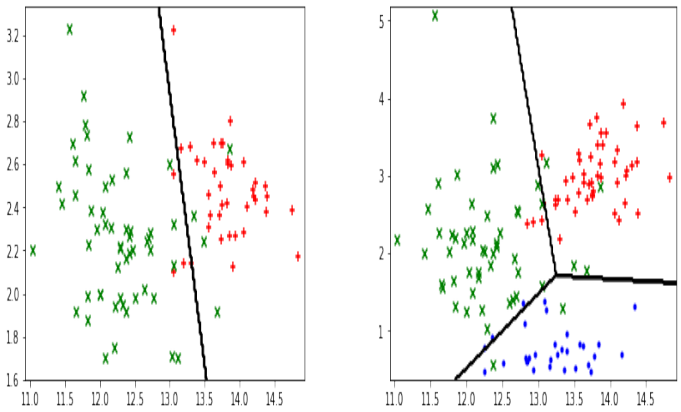


1.4.1 Linear models: decision boundaries given by hyper-planes

This is a demo for logistic regression classification for binary and multi-classed cases. The original data sets would be



Then the decision boundary for logistic regression models would be:



1.4.2 How to find these hyperplanes: logistic regressions

For a collection of subsets $A_1, ..., A_k \subset \mathbb{R}^d$, try to find

$$W = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix} \in \mathbb{R}^{k\times d}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \in \mathbb{R}^{k\times d}, \tag{1.12}$$

such that, for each $1 \le i \le k$ and $j \ne i$

$$(Wx + b)_i > (Wx + b)_j, \ \forall x \in A_i, \tag{1.13}$$

or

$$w_i x + b_i > w_j x + b_j, \ \forall x \in A_i. \tag{1.14}$$

More details of logistic regression will be discussed later.

2

Linear Machine Learning Models

In this chapter, we will mainly consider one statistic model, namely logistic regression, to define classifier for linearly separable sets. In the next chapter, we will study another statistical model, namely support vector machine (SVM). These two linear models form the foundation of deep learning, since the final fully connected output layer of a deep neural network is often give by one of these two linear classifiers. The general intuition is that linear classification models will work well when the different classes are approximately linearly separable. This assumption is made explicit for the SVM model, but the situation is a bit more subtle with the logistic regression.

2.1 Definition of linearly separable sets

In this section, we consider a special class of separable sets, namely linearly separable sets. Let us formally introduce the following definition.

2.1.1 Binary classification

For $k = 2$, there is a very simple geometric interpretation of two linearly separable sets.

Definition 1. *The two sets $A_1, A_2 \subset \mathbb{R}^d$ are linearly separable if there exists a hyperplane*

$$H_0 = \{x : wx + b = 0\}, \tag{2.1}$$

such that $wx + b > 0$ if $x \in A_1$ and $wx + b < 0$ if $x \in A_2$.

Lemma 1. *The two sets $A_1, A_2 \subset \mathbb{R}^d$ are linearly separable if there exists a hyperplane linearly separable if there exists*

$$W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \in \mathbb{R}^{2\times d}, b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \in \mathbb{R}^{2\times d}, \tag{2.2}$$

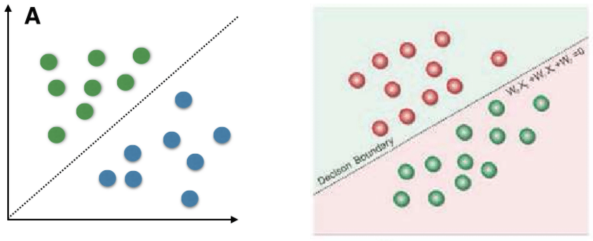


Fig. 2.1. Two linearly separable sets

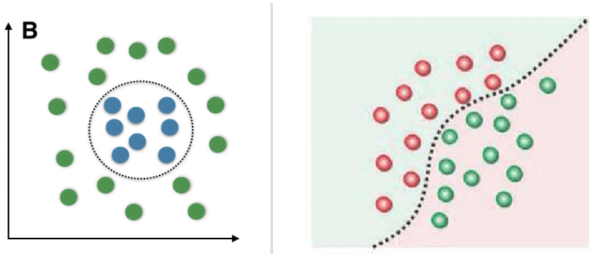


Fig. 2.2. Two Non-linearly separable sets

such that, for each $1 \le i \le 2$ *and* $j \neq i$

$$w_1 x + b_1 > w_2 x + b_2, \ \forall x \in A_1, \tag{2.3}$$

and

$$w_1 x + b_1 < w_2 x + b_2, \ \forall x \in A_2. \tag{2.4}$$

Proof. Here, we can just take $w = w_1 - w_2$ and $b = b_1 - b_2$, then we can check that the hyperplane $wx + b$ satisfies the definition as presented before. □

2.1.2 Multi-class classification

To begin with the definition, let us assume that the data space is divided into k classes represented by k disjoint sets $A_1, A_2, \cdots, A_k \subset \mathbb{R}^d$, which means

$$A = A_1 \cup A_2 \cup \cdots \cup A_k, \ A_i \cap A_j = \emptyset, \forall i \neq j. \tag{2.5}$$

Definition 2 (Linearly Separable). *A collection of subsets* $A_1, ..., A_k \subset \mathbb{R}^d$ *are linearly separable if there exist*

$$W = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix} \in \mathbb{R}^{k \times d}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \in \mathbb{R}^{k \times d}, \tag{2.6}$$

such that, for each $1 \le i \le k$ *and* $j \neq i$

$$(Wx + b)_i > (Wx + b)_j, \ \forall x \in A_i, \tag{2.7}$$

or

$$w_i x + b_i > w_j x + b_j, \ \forall x \in A_i. \tag{2.8}$$

2.1.3 Geometric interpretation for multi-label cases ($k > 2$)

The geometric interpretation for linearly separable sets is less obvious when $k > 2$.

Lemma 2. *Assume that* $A_1, ..., A_k$ *are linearly separable and* $W \in \mathbb{R}^{k\times d}$ *and* $b \in \mathbb{R}^k$ *satisfy* (2.6). *Define*

$$\Gamma_i(W, b) = \{x \in \mathbb{R}^d : (Wx + b)_i > (Wx + b)_j, \ \forall j \neq i\} \tag{2.9}$$

Then for each i,

$$A_i \subset \Gamma_i(W, b) \tag{2.10}$$

We note that each $\Gamma_i(W, b)$ is a polygon whose boundary consists of hyperplanes

$$H_{ij} = \{(w_i - w_j) \cdot x + (b_i - b_j) = 0\}, \quad \forall j \neq i. \tag{2.11}$$

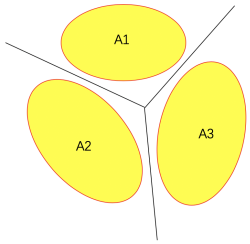


Fig. 2.3. Linearly separable sets in 2-d space (k = 3)

2.1.4 Two more definitions of linearly separable sets

We next introduce two more definitions of linearly separable sets that have more clear geometric interpretation.

Definition 3 (All-vs-One Linearly Separable). *A collection of subsets* $A_1, ..., A_k \subset \mathbb{R}^d$ *is all-vs-one linearly separable if for each* $i = 1, ..., k$, A_i *and* $\cup_{j\neq i} A_j$ *are linearly separable.*

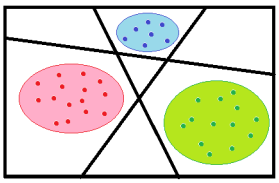


Fig. 2.4. All-vs-One linearly separable sets (k = 3)

Definition 4 (Pairwise Linearly Separable). *A collection of subsets* $A_1, ..., A_k \subset \mathbb{R}^d$ *is pairwise linearly separable if for each pair of indices* $1 \le i < j \le k$, A_i *and* A_j *are linearly separable.*

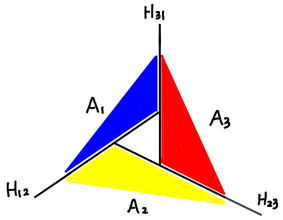


Fig. 2.5. Pairwise linearly separable sets in 2-d space (k = 3)

2.1.5 Comparison of different definitions of linearly separable sets

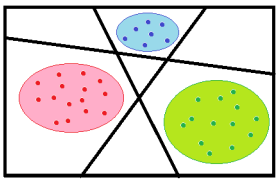


Fig. 2.6. All-vs-One linearly separable sets (k = 3)

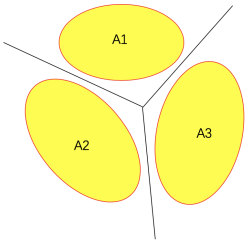


Fig. 2.7. Linearly separable sets in 2-d space (k = 3)

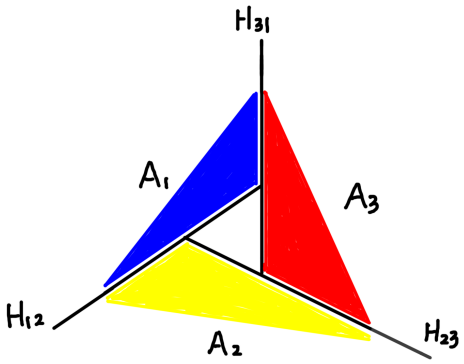


Fig. 2.8. Pairwise separable but not linearly separable sets

We begin by comparing our notion of linearly separable to the two other previously introduced geometric definitions of all-vs-one linearly separable and pairwise lin-

eaerly separable. Obviously, in the case of two classes, they are all equivalent, however, with more than two classes this is no longer the case. We do have the following implications, though.

Lemma 3. *If $A_1, ..., A_k \subset \mathbb{R}^d$ are all-vs-one linearly separable, then they are linearly separable as well.*

Proof. Assume that $A_1, ..., A_k$ are all-vs-one linearly separable. For each i, let w_i, b_i be such that $w_i x + b_i$ separates A_i from $\cup_{j\neq i} A_j$, i.e. $w_i x + b_i > 0$ for $x \in A_i$ and $w_i x + b_i < 0$ for $x \in \cup_{j\neq i} A_j$.

Set $W = (w_1^T, w_2^T, \cdots, w_k^T)^T$, $b = (b_1, b_2, \cdots, b_k)^T$ and observe that if $x \in A_i$, then $(Wx + b)_i > 0$ while $(Wx + b)_j < 0$ for all $j \neq i$. □

Lemma 4. *If $A_1, ..., A_k \subset \mathbb{R}^n$ are linearly separable, then they are pairwise linearly separable as well.*

Proof. If $A_1, ..., A_k \subset \mathbb{R}^d$ are linearly separable, suppose that $W = (w_1^T, w_2^T, \cdots, w_k^T)^T$, $b = (b_1, b_2, \cdots, b_k)^T$. So we have

$$\begin{cases} w_i x + b_i > w_j x + b_j & x \in A_i \\ w_i x + b_i < w_j x + b_j & x \in A_j \end{cases} \tag{2.12}$$

Take $w_{i,j} = w_i - w_j, b_{i,j} = b_i - b_j$, then we have

$$w_{i,j} x + b_{i,j} \begin{cases} > 0 & x \in A_i \\ < 0 & x \in A_j \end{cases} \tag{2.13}$$

So $A_1, ..., A_k$ are pairwise linearly separable. □

However, the converses of both of these statements are false, as the following examples show.

Example 1 (Linearly separable but not all-vs-one linearly separable). Consider the sets $A_1, A_2, A_3 \subset \mathbb{R}$ given by $A_1 = [-4, -2]$, $A_2 = [-1, 1]$, and $A_3 = [2, 4]$. These sets are clearly not one-vs-all linearly separable because A_2 cannot be separated from both A_1 and A_3 by a single plane (in $\mathbb{R}$ this is just cutting the real line at a given number, and A_2 is in the middle).

However, these sets are linearly separated by $W = [-2, 0, 2]^T$ and $b = [-3, 0, -3]^T$, for example.

Example 2 (Pairwise linearly separable but not linearly separable). Consider the sets $A_1, A_2, A_3 \subset \mathbb{R}^2$ shown in figure 2.9. Note that A_i and A_j are separated by hyperplane $H_{i,j}$ (drawn in the figure) and so these sets are pairwise linearly separable. We will show that they are not linearly separable.

Assume to the contrary that $W \in \mathbb{R}^{3\times 2}$ and $b \in \mathbb{R}^2$ separates A_1, A_2, and A_3. Then $(w_i - w_j)x + (b_i - b_j)$ must be a plane which separates A_i and A_j. Now consider the point z in figure 2.9. We see from the figure that given any plane separating A_1 from A_2, z must be on the same side as A_2, given any plane separating A_2 from A_3, z must

be on the same side as A_3, and given any plane separating A_3 from A_1, z must be on the same side as A_1.

This means that $(w_2 - w_1)z + (b_2 - b_1) > 0$, $(w_3 - w_2)z + (b_3 - b_2) > 0$, and $(w_1 - w_3)z + (b_1 - b_3) > 0$. Adding these together, we obtain $0 > 0$, a contradiction.

The essence behind this example is that although the sets A_1, A_2, and A_3 are pairwise linearly separable, no possible pairwise separation allows us to consistently classify arbitrary new points. However, a linear separation would give us a consistent scheme for classifying new points.

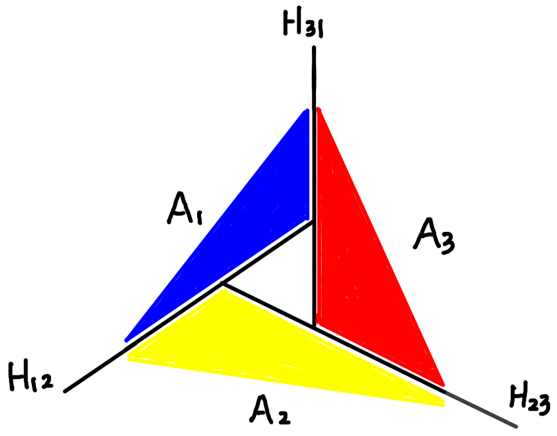


Fig. 2.9. Pairwise separable but not linearly separable sets

So the notion of linear separability is sandwiched in between the more intuitive notions of all-vs-one and pairwise separability. It turns out that linear separability is the notion which is most useful for the k-class classification problem and so we focus on this notion of separability from now on.

2.2 Introduction to logistic regression

2.2.1 Plain logistic regression

We first introduce the next definition of the set of linearly classifiable weights.

Definition 5 (the set of linearly classifiable weights). *Assume that we are given k linearly separable sets $A_1, A_2, \cdots, A_k \in \mathbb{R}^d$, we define the set of classifiable weights as*

$$\boldsymbol{\Theta} = \{\boldsymbol{\theta} = (W, b) : w_i x + b_i > w_j x + b_j,\ \forall x \in A_i,\ j \neq i, i = 1, \cdots, k\} \tag{2.14}$$

which means those (W, b) can separate $A_1, A_2, \cdots, A_k$ absolutely correctly.

Our linearly separable assumption implies that $\boldsymbol{\Theta} \neq \emptyset$. Now we know the existence of linearly classifiable weights. But how can we find one element in $\boldsymbol{\Theta}$?

Definition 6 (soft-max). *Given parameter $\boldsymbol{\theta} = (W, b)$, a soft-max mapping $\boldsymbol{p} : \mathbb{R}^d \to \mathbb{R}^k$ is a mapping with the following formulation*

$$\boldsymbol{p}(x; \boldsymbol{\theta}) = \frac{e^{Wx+b}}{e^{Wx+b} \cdot \mathbf{1}} = \frac{1}{\sum_{i=1}^{k} e^{w_i x + b_i}} \begin{pmatrix} e^{w_1 x + b_1} \\ e^{w_2 x + b_2} \\ \vdots \\ e^{w_k x + b_k} \end{pmatrix} = \begin{pmatrix} p_1(x; \boldsymbol{\theta}) \\ p_2(x; \boldsymbol{\theta}) \\ \vdots \\ p_k(x; \boldsymbol{\theta}) \end{pmatrix} \tag{2.15}$$

where $e^{Wx+b} = \begin{pmatrix} e^{w_1 x + b_1} \\ e^{w_2 x + b_2} \\ \vdots \\ e^{w_k x + b_k} \end{pmatrix}$, $\mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^k$, *and the i-th component*

$$p_i(x; \boldsymbol{\theta}) = \frac{e^{w_i x + b_i}}{\sum_{i=1}^{k} e^{w_i x + b_i}}. \tag{2.16}$$

The soft-max mapping have several important properties.

1. $0 < p_i(x; \boldsymbol{\theta}) < 1,\ \sum_i p_i(x; \boldsymbol{\theta}) = 1$.
2. $p_i(x; \boldsymbol{\theta}) > p_j(x; \boldsymbol{\theta}) \Leftrightarrow w_i x + b_i > w_j x + b_j$. This implies that the linearly classifiable weights have an equivalent description as

$$\boldsymbol{\Theta} = \left\{\boldsymbol{\theta} : p_i(x; \boldsymbol{\theta}) > p_j(x; \boldsymbol{\theta}),\ \forall x \in A_i,\ j \neq i, i = 1, \cdots, k\right\} \tag{2.17}$$

3. We usually use the max-out method to do classification. For a given data point x, we first use a soft-max mapping to map it to $\boldsymbol{p}(x; \boldsymbol{\theta})$, then we attached x to the class $i = \arg\max_j p_i(x; \boldsymbol{\theta})$.

Remark 1. The first property implies that $\boldsymbol{p}(x;\boldsymbol{\theta})$ can be regarded as a probability distribution of data points which means given $x \in \mathbb{R}^d$, we have $x \in A_i$ with probability $p_i(x;\boldsymbol{\theta})$, $i = 1, \cdots, k$.

The last properties means we pick the label i as the class of x such that $x \in A_i$ has the biggest probability $p_i(x;\boldsymbol{\theta})$.

More detailed discussion of logistic regression from the probability perspective will be presented in the nearly future.

From the above properties, we can define the next likelihood function to help find elements in Θ:

$$P(\boldsymbol{\theta}) = \prod_{i=1}^{k} \prod_{x \in A_i} p_i(x;\boldsymbol{\theta}) \tag{2.18}$$

where this likelihood function comes from its probabilistic interpretation which we will discuss later. Based on the property that

$$p_i(x;\boldsymbol{\theta}) = \max_{1 \le j \le k} p_j(x;\boldsymbol{\theta}), \forall x \in A_i, \tag{2.19}$$

if $\boldsymbol{\theta} \in \Theta$. This somehow means that if

$$P(\boldsymbol{\theta}) = \prod_{i=1}^{k} \prod_{x \in A_i} p_i(x;\boldsymbol{\theta}) = \max, \tag{2.20}$$

if $\boldsymbol{\theta} \in \Theta$. Or we say that we may use the next optimization problem

$$\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta}). \tag{2.21}$$

to find an element in Θ.

More precisely, let us introduce the next lemmas (properties) of $P(\boldsymbol{\theta})$.

Lemma 5. *Assume that the sets $A_1, A_2, \cdots, A_k$ are linearly separable. Then we have*

$$\left\{\boldsymbol{\theta} : \ P(\boldsymbol{\theta}) > \frac{1}{2}\right\} \subset \boldsymbol{\Theta}. \tag{2.22}$$

Proof. It suffices to show that if $\boldsymbol{\theta} \notin \boldsymbol{\Theta}$, we must have $P(\boldsymbol{\theta}) \leq \frac{1}{2}$. For any $\boldsymbol{\theta} \notin \boldsymbol{\Theta}$, there must exist an i_0 ,an $x_0 \in A_{i_0}$ and a $j_0 \neq i_0$ such that

$$w_{i_0} x_0 + b_{i_0} \leq w_{j_0} x_0 + b_{j_0}. \tag{2.23}$$

Then we have

$$p_{i_0}(x_0; \boldsymbol{\theta}) \leq \frac{e^{w_{i_0} x0 + b_{i_0}}}{e^{w_{i_0} x_0 + b_{i_0}} + e^{w_{j_0} x_0 + b_{j_0}}} \leq \frac{1}{2}. \tag{2.24}$$

Notice that $p_i(x; \boldsymbol{\theta}) < 1$ for all $i = 1, \cdots, k$, $x \in A$. So

$$P(\boldsymbol{\theta}) < p_{i_0}(x_0; \boldsymbol{\theta}) \leq \frac{1}{2}. \tag{2.25}$$

□

Lemma 6. *If $A_1, A_2, \cdots, A_k$ are linearly separable and $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, we have*

$$\lim_{\alpha \to +\infty} p_i(x; \alpha\boldsymbol{\theta}) = 1 \Leftrightarrow x \in A_i. \tag{2.26}$$

Proof. We first note that if $x \in A_i$,

$$p_i(\theta, x) = \frac{1}{1 + \sum\limits_{j \neq i} e^{\alpha[(w_j x + b_j) - (w_i x + b_i)]}} \to 1, \quad \text{as} \quad \alpha \to \infty. \tag{2.27}$$

On the other hand, if $x \notin A_i$,

$$p_i(x; \boldsymbol{\alpha\theta}) = \frac{1}{1 + \sum\limits_{j \neq i} e^{\alpha[(w_j x + b_j) - (w_i x + b_i)]}} \leq \frac{1}{2}. \tag{2.28}$$

This implies that if $x \notin A_i$, $\lim_{\alpha \to \infty} p_i(x; \alpha\boldsymbol{\theta}) \neq 1$ which is equivalent to the proposition that if $\lim_{\alpha \to \infty} p_i(x; \alpha\boldsymbol{\theta}) = 1$, then $x \in A_i$. □

Lemma 7. *If $A_1, A_2, \cdots, A_k$ are linearly separable,*

$$\boldsymbol{\Theta} = \left\{\boldsymbol{\theta} : \lim_{\alpha \to +\infty} P(\alpha\boldsymbol{\theta}) = 1\right\}. \tag{2.29}$$

Proof. We first note that if $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, we have $\lim_{\alpha \to +\infty} p_i(x; \alpha\boldsymbol{\theta}) = 1$ for all $x \in A_i$. So

$$\lim_{\alpha\to+\infty} H(\alpha\boldsymbol{\theta}) = \lim_{\alpha\to+\infty} \prod_{i=1}^{k} \prod_{x\in A_i} p_i(x;\alpha\boldsymbol{\theta}) = \prod_{i=1}^{k} \prod_{x\in A_i} \lim_{\alpha\to+\infty} p_i(x;\alpha\boldsymbol{\theta}) = 1. \tag{2.30}$$

On the other hand, if $\lim_{\alpha\to+\infty} P(\alpha\boldsymbol{\theta}) = 1$, there must exist one $\alpha_0 > 0$ such that $P(\alpha_0\boldsymbol{\theta}) > \frac{1}{2}$. From Lemma 5, we have $\alpha_0\boldsymbol{\theta} \in \boldsymbol{\Theta}$, which means $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. □

These properties above imply that if we can obtain a classifiable weight through maximizing $P(\boldsymbol{\theta})$, while lemma 7 tells us that $P(\boldsymbol{\theta})$ will not have a minimum actually.

More specifically, we just need to find some $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ such that

$$P(\boldsymbol{\Theta}) > \frac{1}{2} \Leftrightarrow L(\boldsymbol{\theta}) := -\log P(\boldsymbol{\theta}) < \log(2). \tag{2.31}$$

Question: how to find these element?

2.2.2 Regularized logistic regression

Here, we start from the regularization term $e^{-\lambda R(\|\boldsymbol{\theta}\|)}$ with these next properties:

1. $\lambda > 0$.
2. $R(t)$ is a strictly increasing function on $\mathbb{R}^+$ with $R(0) = 0$, $\lim_{t\to+\infty} R(t) = +\infty$. For example, $R(t) = t^2$.
3. $\|\cdot\|$ is a norm on $R^{k\times(d+1)}$, a commonly used norm is the following Frobenius norm:

$$\|\boldsymbol{\theta}\|_F = \sqrt{\sum_{i,j} W_{ij}^2 + \sum_i b_i^2}. \tag{2.32}$$

Based on this regularization term, we may consider the following regularized likelihood function $P_\lambda(\boldsymbol{\theta})$ as

$$P_\lambda(\boldsymbol{\theta}) = P(\boldsymbol{\theta}) e^{-\lambda R(\|\boldsymbol{\theta}\|)}. \tag{2.33}$$

Here, let us define

$$\boldsymbol{\Theta}_\lambda = \arg\max_{\boldsymbol{\theta}} P_\lambda(\boldsymbol{\theta}), \tag{2.34}$$

where we have the next definition of arg max

$$\arg\max_{\boldsymbol{\theta}} P_\lambda(\boldsymbol{\theta}) = \left\{ \boldsymbol{\theta} \ : \ P_\lambda(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} P_\lambda(\boldsymbol{\theta}) \right\}. \tag{2.35}$$

The next lemma show that the maximal set of modified objective is not empty.

Lemma 8. *Suppose that $A_1, A_2, \cdots, A_k$ are linearly separable, then*

1. *if $\lambda = 0$, $\boldsymbol{\Theta}_0 = \emptyset$,*
2. *$\boldsymbol{\Theta}_\lambda$ must be nonempty for all $\lambda > 0$.*

Proof. Lemma 7 shows the first proposition. For the second proposition, we notice that

1. $P_\lambda(\mathbf{0}) = \frac{1}{k^N}$.
2. $\exists\, M_\lambda > 0$ such that $e^{-\lambda R(\|\boldsymbol{\theta}\|)} < \frac{1}{k^N}$ whenever $\|\boldsymbol{\theta}\| > M_\lambda$ because of the properties of $R(\|\boldsymbol{\theta}\|)$.

So a maxima on $\{\boldsymbol{\theta} : \|\boldsymbol{\theta}\| \le M_\lambda\}$ must be a global maxima. Then we can easily obtain the result in the lemma from the boundedness and closeness of $\{\boldsymbol{\theta} : \|\boldsymbol{\theta}\| \le M_\lambda\}$. □

Furthermore, we have the next theorem which shows that we can indeed get Θ by maximizing $P_\lambda(\boldsymbol{\theta})$.

Theorem 1. *If $A_1, A_2, \cdots, A_k$ are linearly separable,*

$$\boldsymbol{\Theta}_\lambda \subset \boldsymbol{\Theta}, \tag{2.36}$$

when $\lambda > 0$ and sufficiently small.

Proof. By Lemma 5, we can take $\boldsymbol{\theta}_0 \in \boldsymbol{\Theta}$ such that $P(\boldsymbol{\theta}_0) > \frac{3}{4}$. Then, for any $\lambda < \frac{\log \frac{3}{2}}{R(\|\boldsymbol{\theta}_0\|)}$, $\boldsymbol{\theta}_\lambda \in \boldsymbol{\Theta}_\lambda$, we have

$$P(\boldsymbol{\theta}_\lambda) \ge P_\lambda(\boldsymbol{\theta}_\lambda) \ge P_\lambda(\boldsymbol{\theta}_0) = P(\boldsymbol{\theta}_0) e^{-\lambda R(\|\boldsymbol{\theta}_0\|)} > \frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2},$$

which implies that $\boldsymbol{\theta}_\lambda \in \Theta$. Thus, for any $0 < \lambda < \frac{\log \frac{3}{2}}{R(\|\boldsymbol{\theta}_0\|)}$, $\boldsymbol{\Theta}_\lambda \subset \boldsymbol{\Theta}$.

□

The design of logistic regression is that maximize $P_\lambda(\boldsymbol{\theta})$ is equivalent to minimize $-\log P_\lambda(\boldsymbol{\theta})$, i.e.,

$$\max_{\boldsymbol{\theta}} \{P_\lambda(\boldsymbol{\theta})\} \Leftrightarrow \min_{\boldsymbol{\theta}} \{-\log P_\lambda(\boldsymbol{\theta})\}, \tag{2.37}$$

while the second one is more convenient to evaluate the gradient. Meanwhile, we add a regularization term $R(\boldsymbol{\theta})$ to the objective function which makes the optimization problem has a unique solution.

Mathematically, we can formulate Logistic regression as

$$\min_{\boldsymbol{\theta}} L_\lambda(\boldsymbol{\theta}), \tag{2.38}$$

where

$$L_\lambda(\boldsymbol{\theta}) := -\log P_\lambda(\boldsymbol{\theta}) = -\log P(\boldsymbol{\theta}) + \lambda R(\|\boldsymbol{\theta}\|) = L(\boldsymbol{\theta}) + \lambda R(\|\boldsymbol{\theta}\|), \tag{2.39}$$

with

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{k} \sum_{x \in A_i} \log p_i(x; \boldsymbol{\theta}). \tag{2.40}$$

Then we have the next logistic regression algorithm.

Algorithm 1 Logistic Regression

Given data $A_1, A_2, \cdots, A_k$, find

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_\lambda(\boldsymbol{\theta}), \tag{2.41}$$

for some sufficient small $\lambda > 0$.

Remark 2. Here

$$L(\boldsymbol{\theta}) = -\log P(\boldsymbol{\theta}), \tag{2.42}$$

is known as the loss function of logistic regression model. The next reasons may show that why $L(\boldsymbol{\theta})$ is popular.

1. It is more convenient to take gradient for $L(\boldsymbol{\theta})$ than $P(\boldsymbol{\theta})$.
2. $L(\boldsymbol{\theta})$ is related the so-called cross-entropy loss function which will be discussed in the next section.
3. $L(\boldsymbol{\theta})$ is a convex function which will be discussed later.

2.3 KL divergence, cross-entropy and logistic regression

KL divergence and cross-entropy

Given two discrete probability distributions,

$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_k \end{pmatrix}, q = \begin{pmatrix} q_1 \\ \vdots \\ q_k \end{pmatrix}$$

namely $0 \le p_i, q_i \le 1$ and $\sum_{i=1}^k p_i = \sum_{i=1}^k q_i = 1$. The KL divergence defines a special distance between p and q:

$$D_{\mathrm{KL}}(q, p) = \sum_{i=1}^{k} q_i \log \frac{q_i}{p_i}. \tag{2.43}$$

$D_{\mathrm{KL}}(q, p)$ works like a "distance" without the symmetry:

Lemma 9.

1. $D_{\mathrm{KL}}(q, p) \ge 0$;
2. $D_{\mathrm{KL}}(q, p) = 0$ *if and only if* $p = q$;

Proof. We first note that the elementary inequality

$$\log x \le x - 1, \quad \text{for any } x \ge 0, \tag{2.44}$$

and the equality holds if and only if $x = 1$.

$$-D_{\mathrm{KL}}(q,p) = -\sum_{i=1}^{c} q_i \log \frac{q_i}{p_i} = \sum_{i=1}^{k} q_i \log \frac{p_i}{q_i} \le \sum_{i=1}^{k} q_i (\frac{p_i}{q_i} - 1) = 0. \tag{2.45}$$

And the equality holds if and only if

$$\frac{p_i}{q_i} = 1 \quad \forall i = 1 : k. \tag{2.46}$$

□

Note that

$$D_{\mathrm{KL}}(q,p) = \sum_{i=1}^{k} q_i \log \frac{q_i}{p_i} = \sum_{i=1}^{k} q_i \log q_i - \sum_{i=1}^{k} q_i \log p_i \tag{2.47}$$

We write

$$H(q,p) = H(q) + D_{\mathrm{KL}}(q,p), \tag{2.48}$$

where

$$H(q) = -\sum_{i=1}^{k} q_i \log q_i, \tag{2.49}$$

which is called entropy for distribution p and

$$H(q,p) = -\sum_{i=1}^{k} q_i \log p_i. \tag{2.50}$$

which is called cross-entropy for distribution p and q.

It follows from the relation (2.48) that

$$\arg\min_p D_{\mathrm{KL}}(q,p) = \arg\min_p H(q,p). \tag{2.51}$$

Cross-entropy

In §2.3 we introduced the concept of cross-entropy, which can be used to define a loss function in machine learning and optimization. Let us assume y_i is the true label for x_i, for example $y_i = e_{k_i}$ if $x_i \in A_{k_i}$. Then ,consider the predicted distribution

$$\boldsymbol{p}(x;\boldsymbol{\theta}) = \frac{1}{\sum_{i=1}^{k} e^{w_i x + b_i}} \begin{pmatrix} e^{w_1 x + b_1} \\ e^{w_2 x + b_2} \\ \vdots \\ e^{w_k x + b_k} \end{pmatrix} = \begin{pmatrix} p_1(x;\boldsymbol{\theta}) \\ p_2(x;\boldsymbol{\theta}) \\ \vdots \\ p_k(x;\boldsymbol{\theta}) \end{pmatrix} \tag{2.52}$$

for any data $x \in A$. By (2.51), we have

$$\arg\min_{\theta} \sum_{i=1}^{N} D_{\mathrm{KL}}(y_i, \boldsymbol{p}(x_i;\boldsymbol{\theta})) = \arg\min_{\theta} \sum_{i=1}^{N} H(y_i, \boldsymbol{p}(x_i;\boldsymbol{\theta})), \tag{2.53}$$

Recall that we have all data $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$. Then, it is natural to consider the loss function as following:

$$\sum_{j=1}^{N} H(y_i, \boldsymbol{p}(x_i;\boldsymbol{\theta})), \tag{2.54}$$

which measures the distance between the real label and predicted one for all data. In the meantime, we can check that

$$\begin{aligned}
\sum_{j=1}^{N} H(y_j, \boldsymbol{p}(x_j;\boldsymbol{\theta})) &= -\sum_{j=1}^{N} y_j \cdot \log \boldsymbol{p}(x_j;\boldsymbol{\theta}) \\
&= -\sum_{j=1}^{N} \log p_{i_j}(x_i;\boldsymbol{\theta}) \quad (\text{because } y_j = e_{i_j} \text{ for } x_j \in A_{i_j}) \\
&= -\sum_{i=1}^{k} \sum_{x\in A_i} \log p_i(x;\boldsymbol{\theta}) \\
&= -\log \prod_{i=1}^{k} \prod_{x\in A_i} p_i(x;\boldsymbol{\theta}) \\
&= L(\theta)
\end{aligned} \tag{2.55}$$

That is to say, the logistic regression loss function defined by likelihood is exact the loss function defined by measuring the distance between real label and predicted one via cross-entropy. Or we can note as

$$\min_{\boldsymbol{\theta}} L_\lambda(\boldsymbol{\theta}) \Leftrightarrow \min_{\boldsymbol{\theta}} \sum_{j=1}^{N} H(y_i, \boldsymbol{p}(x_i;\boldsymbol{\theta})) + \lambda R(\|\boldsymbol{\theta}\|) \Leftrightarrow \min_{\boldsymbol{\theta}} \sum_{j=1}^{N} D_{\mathrm{KL}}(y_i, \boldsymbol{p}(x_i;\boldsymbol{\theta})) + \lambda R(\|\boldsymbol{\theta}\|). \tag{2.56}$$

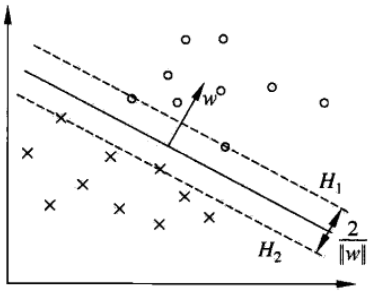
2.4 Binary LR and SVM and their relations

Given a binary linealy separable classification dataset $(x_i, y_i)_{i=1}^N$, where $x_i \in \mathbb{R}^d, y_i \in \{-1, +1\}$. We use A_1, A_2 to denote the data with label $+1, -1$ respectively. Our goal is to find a $\theta = (w, b)$ where $w \in \mathbb{R}^{1\times d}, b \in \mathbb{R}$ such that the hyperplane $H_\theta = \{x : wx + b = 0\}$ can separate A_1, A_2.

2.4.1 Binary SVM

Binary SVM wants to find the classifiable hyperplane which has the biggest distance with A_1 and A_2.

$$\max_{w,b} \frac{\min_i y_i(wx_i + b)}{\|w\|_2} \tag{2.57}$$



Intuitively, the best separating hyperplane H are only determined by those data points who are closest to H. Those data points are called support vector, and this method are called support vector machine.

Without loss of generality, we may restrict the norm of $\|w\|$ to be 1, which leads to a equivalent optimization problem

$$\max_{\|w\|_2=1} \min_i y_i(wx_i + b) \tag{2.58}$$

Actually, we can prove $\operatorname{argmax}_{\|w\|_2=1} \min_i y_i(wx_i + b)$ is nonempty, but here we just admit this fact and only prove the uniqueness of the solution.

Lemma 10. *If A_1, A_2 are linearly separable, then*

$$\operatorname*{argmax}_{\|w\|_2=1} \min_i y_i(wx_i + b) \tag{2.59}$$

is a singleton set.

Proof. Denote $m(w,b) = \min_i y_i(wx_i + b)$. Notice that $m(w,b)$ is a concave homogeneous function w.r.t w, b and $\|\cdot\|_2$ is a strictly convex norm. Suppose there are two solution (w_1, b_1) and (w_2, b_2) such that $w_1 \neq w_2$, take $\overline{w} = \frac{w_1+w_2}{2}, \overline{b} = \frac{b_1+b_2}{2}$, we must have

$$m(\overline{w}, \overline{b}) \geq \frac{m(w_1,b_1) + m(w_2,b_2)}{2} = \max_{\|w\|_2=1} m(w,b), \tag{2.60}$$

and

$$\|\overline{w}\|_2 < 1. \tag{2.61}$$

So

$$m(\frac{\overline{w}}{\|\overline{w}\|_2}, \frac{\overline{b}}{\|\overline{w}\|_2}) = \frac{m(\overline{w}, \overline{b})}{\|\overline{w}\|_2} > \max_{\|w\|_2=1} m(w,b), \tag{2.62}$$

which leads to a contradiction. So all the solution must have the same w, we denote it as w^*. Then if (w^*, b^*) is a solution of problem (2.59), we must have

$$b^* \in \operatorname*{argmax}_b m(w^*, b) \tag{2.63}$$

Actually,

$$m(w^*, b) = \min\{b + \min_{x\in A_1} w^* x, -b + \min_{x\in A_2}(-w^* x)\}, \tag{2.64}$$

easy to observe that $\operatorname{argmax}_b m(w^*, b)$ is a singleton set and

$$b^* = \frac{\min_{x\in A_2}(-w^* x) - \min_{x\in A_1} w^* x}{2}. \tag{2.65}$$

□

Denote

$$\theta^*_{SVM} = (w^*_{SVM}, b^*_{SVM}) = \operatorname*{argmax}_{\|w\|=1} \min_i y_i(wx_i + b). \tag{2.66}$$

Theorem 2 (Representation Theorem). *w^*_{SVM} must be a linear combination of $x_i^T, i = 1, 2, \cdots, N$.*

Proof. Denote

$$S = \operatorname{span}\{x_i^T\}_{i=1}^N \tag{2.67}$$

Then we have

$$\mathbb{R}^{1\times d} = S \oplus^\perp S^\perp \tag{2.68}$$

So w^*_{SVM} can be uniquely decomposed as $w^*_{SVM} = w^*_S + w^*_{S^\perp}$ where $w_S \in S$ and $w^*_{S^\perp} \in S^\perp$. We will prove that $w^*_{S^\perp} = 0$. Suppose not, we have

$$\|w_S^*\|_2 < \|w^*\|_2 = 1. \tag{2.69}$$

Notice that

$$w_{SVM}^* x_i = w_S^* x_i, \ \forall i = 1, 2, \cdots, N. \tag{2.70}$$

Thus we have

$$\min_i y_i(w_{SVM}^* x_i + b^*) = \min_i y_i(w_S^* x_i + b^*) \tag{2.71}$$

So

$$\min_i y_i(w_{SVM}^* x_i + b_{SVM}^*) < \frac{\min_i y_i(w_S^* x_i + b_{SVM}^*)}{\|w_S^*\|} = \min_i y_i(\frac{w_S^*}{\|w_S^*\|_2} x_i + \frac{b_{SVM}^*}{w_S^*}), \tag{2.72}$$

which leads to a contradiction to the definition of θ_{SVM}^*. □

2.4.2 Soft margin maximization and kernel methods

We may rewrite the SVM problem as

$$\min_{w,b} \|w\|^2, \tag{2.73}$$

$$s.t.\ y_i(wx_i + b) \geq 1, \ \forall i. \tag{2.74}$$

Notice that the feasible domain of margin maximization is nonempty if and only if dataset is linearly separable. So when the data is linearly nonseparable, this method can't even get a classifier even though it may not be good. One way to handle this problemis to relax the constraint by adding relaxation variables. The following problem is called soft margin maximization

$$\min_{w,b,\xi} \|w\|^2 + C \sum_{i=1}^{N} \xi_i, \tag{2.75}$$

$$s.t.\ y_i(wx_i + b) + \xi_i \geq 1, \ \forall i. \tag{2.76}$$

$$\xi_i \geq 0. \tag{2.77}$$

where $C > 0$. The above problem is equivalent to

$$\min_{w,b} \|w\|^2 + C \sum_{i=1}^{N} \mathrm{ReLU}(1 - y_i(wx_i + b)) \tag{2.78}$$

Or we denote $\lambda = \frac{1}{C}$, it can be reformulated as

$$\min_{w,b} \sum_{i=1}^{N} \mathrm{ReLU}(1 - y_i(wx_i + b)) + \lambda\|w\|^2. \tag{2.79}$$

We can still prove that the solution of (2.79) satisfies the representer theorem. Thus we can restrict w to be in the set S. Assume that

$$w = \sum_{i=1}^{N} \alpha_i x_i^T, \tag{2.80}$$

Denote $\alpha = (\alpha_1, \cdots, \alpha_N)^T$. We can rewrite the problem (2.79) as

$$\min_{\alpha} \sum_{i=1}^{N} \text{ReLU}(1 - y_i(\sum_{j=1}^{N} \langle x_i, x_j \rangle \alpha_j + b)) + \lambda \alpha^T (\langle x_i, x_j \rangle)_{N\times N} \alpha \tag{2.81}$$

We can see that the whole problem is only determined by the inner product of data points but not the data itself directly.

Use the above formulation, we can induce nonlinearity in SVM. Denote the input space as X where $\{x_i\}_{i=1}^N \subset X$. We use two steps to obtain a nonlinear classification model. First, we use a nonlinear feature mapping $\phi : X \to \mathcal{H}$ to map input space X to a feature space $\mathcal{H}$. Second, we use linear SVM to do classification on $\{\phi(x_i)\}_{i=1}^N \subset \mathcal{H}$.

We may just asssume dataset after feature mapping ϕ is linearly separable. Then, the SVM problem after doing feature mapping can be formulated as problem (2.79) as

$$\min_{\alpha} \sum_{i=1}^{N} \text{ReLU}(1 - y_i(\sum_{j=1}^{N} \langle \phi(x_i), \phi(x_j) \rangle \alpha_j + b)) + \lambda \alpha^T (\langle \phi(x_i), \phi(x_j) \rangle)_{N\times N} \alpha \tag{2.82}$$

Notice that to obtain the above problem we don't really need to know what exactly is the nonlinear mapping ϕ, but only need to compute the value of $< \phi(x_i), \phi(x_j) >$. So we define a kernel function $k : X \times X \to \mathbb{R}$ such that

$$k(x, y) = \langle \phi(x), \phi(y) \rangle, \ \ x, y \in X. \tag{2.83}$$

Then the kernel SVM can be formulated as

$$\min_{\alpha} \sum_{i=1}^{N} \text{ReLU}(1 - y_i(\sum_{j=1}^{N} k(x_i, x_j) \alpha_j + b)) + \lambda \alpha^T (k(x_i, x_j))_{N\times N} \alpha \tag{2.84}$$

In practice, we just need to find a proper kernel function instead of a good nonlinear feature mapping. Here we list some common used kernel functions:

- Polynomial kernel: $k(x, y) = (a < x, y > +b)^n, a > 0, b \geq 0, n \in \mathbb{N}^+$.
- Gaussian kernel: $k(x, y) = e^{-\gamma \|x-y\|^2}, \gamma > 0$.
- Laplacian kernel: $k(x, y) = e^{-\gamma \|x-y\|}, \gamma > 0$
- Tanh kernel: $k(x, y) = \tanh(a < x, y > +b), a > 0, b \geq 0$.

2.4.3 Binary Logistic Regression

In multi-class Logistic regression, if we use $\|W\|$ to replace $\|\boldsymbol{\theta}\|$ in regularization term, we can get another version of logistic regression:

$$\mathcal{L}_\lambda(\boldsymbol{\theta}) = -\sum_{i=1}^{k}\sum_{x\in A_i}\log p_i(x;\boldsymbol{\theta}) + \lambda R(\|W\|), \tag{2.85}$$

where $p_i(x;\boldsymbol{\theta})$ and $R(\cdot)$ share the same definitions as in previous notes of logistic regression. Let denote

$$\boldsymbol{\Theta}_\lambda = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_\lambda(\boldsymbol{\theta}). \tag{2.86}$$

Lemma 11. *For any $W \in \mathbb{R}^{k\times d}, b \in \mathbb{R}^k, \alpha \in \mathbb{R}$, we have*

$$\mathcal{L}_\lambda(W, b) = \mathcal{L}_\lambda(W, b + \alpha\mathbf{1}), \tag{2.87}$$

where $\mathbf{1} = (1, 1, \cdots, 1)^T \in \mathbb{R}^k$.

Lemma 12. *If $k = 2$, given any $\boldsymbol{\theta}_\lambda = \begin{pmatrix} w_1 & b_1 \\ w_2 & b_2 \end{pmatrix} \in \boldsymbol{\Theta}_\lambda$, we have*

$$w_1 = -w_2.$$

According to the above two lemmas (homeworks), we can restrict $\boldsymbol{\theta}$ to have the form

$$\boldsymbol{\theta} = \begin{pmatrix} \frac{w}{2} & \frac{b}{2} \\ -\frac{w}{2} & -\frac{b}{2} \end{pmatrix}, \tag{2.88}$$

so our score mapping can be written as

$$\boldsymbol{p}(x;\theta) = \begin{pmatrix} \frac{1}{1+e^{-(wx+b)}} \\ \frac{1}{1+e^{wx+b}} \end{pmatrix}. \tag{2.89}$$

Denote $\theta = (w, b)$ where $w \in \mathbb{R}^d, b \in \mathbb{R}$, correspondingly, we have

$$P(\theta) = \prod_{i=1}^{N} \frac{1}{1 + e^{-y_i(wx+b)}}. \tag{2.90}$$

Here we have the new "label" y_i defined by

$$y_i = \begin{cases} 1, & \text{if} \quad x_i \in A_1 \\ -1, & \text{if} \quad x_i \in A_2 \end{cases}. \tag{2.91}$$

Thus we have

$$L(\theta) = -\log P(\theta) = \sum_{i=1}^{N} -\log(1 + e^{-y_i(wx+b)}), \tag{2.92}$$

and take $R(t) = t^2$, we have

$$\mathcal{L}_\lambda(\theta) = L(\theta) + \lambda\|w\|_2^2 = \sum_{i=1}^{N} -\log(1 + e^{-y_i(wx+b)}) + \lambda\|w\|_2^2. \tag{2.93}$$

Lemma 13 (optional homework). *Assume that A_1, A_2 are linearly separable, and we follow the same definition of linearly classifiable weights:*

$$\boldsymbol{\Theta} = \left\{\boldsymbol{\theta} : p_i(x;\boldsymbol{\theta}) > p_j(x;\boldsymbol{\theta}),\ \forall x \in A_i, j \neq i, i = 1, 2\right\}, \tag{2.94}$$

where

$$\boldsymbol{p}(x;\theta) = \begin{pmatrix} \frac{1}{1+e^{-(wx+b)}} \\ \frac{1}{1+e^{wx+b}} \end{pmatrix}. \tag{2.95}$$

Then, we have the following statement:

1. *$\theta = (w, b) \in \boldsymbol{\Theta}$ if and only if*

$$\frac{1}{1 + e^{-y_i(wx+b)}} > \frac{1}{2},\ \forall i = 1, 2\cdots, N.$$

2. *If $P(\theta) > \frac{1}{2}$, then θ must be classifiable, i.e.*

$$\{\theta : P(\theta) > \frac{1}{2}\} \subset \boldsymbol{\Theta}.$$

3. *Prove that*

$$\boldsymbol{\Theta} = \{\theta : \lim_{\alpha\to+\infty} P(\alpha\theta) = 1\}.$$

Lemma 14. *$L(\theta)$ is a strictly convex function without any global minima.*

Actually, we can prove $\operatorname{argmin}_{w,b} \mathcal{L}_\lambda(\theta)$ is nonempty for λ sufficiently small, but here we just admit this fact and only prove the uniqueness of the solution.

Lemma 15. *If A_1, A_2 are linearly separable, then*

$$\operatorname*{argmin}_{w,b} \mathcal{L}_\lambda(\theta) \tag{2.96}$$

is a singleton set for λ sufficiently small.

Proof. Because $L(\theta)$ is strictly convex w.r.t. θ and $\|w\|^2$ is convex w.r.t. θ, so $\mathcal{L}(\theta, \lambda) = L(\theta) + \lambda\|w\|_2^2$ is stricly convex w.r.t. θ, which implies our result directly. □

For λ sufficiently small, denote

$$\theta_{LR}(\lambda) = (w_{LR}(\lambda), b_{LR}(\lambda)) = \operatorname*{argmin}_{w,b} \mathcal{L}_\lambda(\theta). \tag{2.97}$$

Theorem 3. *If A_1, A_2 are linearly separable, then $\frac{\theta_{LR}(\lambda)}{\|w_{LR}(\lambda)\|}$ converge to θ^*_{SVM} as $\lambda \to 0$, i.e.*

$$\theta^*_{SVM} = \lim_{\lambda\to 0} \frac{\theta_{LR}(\lambda)}{\|w_{LR}(\lambda)\|}. \tag{2.98}$$

3

Probability

3.1 Introduction to probability

Introduction to Probability: https://link-springer-com.ezaccess.libraries.psu.edu/book/10.1007%2F978-0-387-21736-9

3.2 Basic probability

3.3 Basic Probability Theory

- Outcome — Set of possible outcomes
- Event — Subset of possible outcomes, an event is something which can happen or not happen
- Distribution — measure on the outcome space, just give the probability of each outcome
- Independence — "Events that are unrelated", depends on the distribution

3.3.1 Discrete Examples

- Coin Flip:
  - Outcomes: $\{H, T\}$
  - Event: If we toss a coin once, there are two possible events: $\{H\}$ and $\{T\}$; If we toss a coin twice, the event that the first toss is heads is $A = \{HH, HT\}$; If we toss a coin three times, the event that the second toss is heads is $B = \{HHH, THT, HHT, THT\}$.
  - Distribution: If we toss a coin, the possibility that it is H and the possibility that it is T are the same, that is $p(\{H\}) = p(\{T\}) = 0.5$.
- Rolling a Die:
  - Outcomes:$\{1, 2, 3, 4, 5, 6\}$

- Event: some examples $E_1 = \{3\}$— rolling a 3 $E_2 = \{1, 3, 5\}$ — rolling an odd number$E_3 = \{2, 4, 6\}$ — rolling an even number$E_4 = \{1, 2\}$— rolling either a 1 or 2
- Distribution: $p(\{1\}) = p(\{2\}) = ... = p(\{6\}) = \frac{1}{6}$(most common case) $p(\{1\}) = \frac{1}{2}, p(\{2\}) = ... = p(\{6\}) = \frac{1}{10}$ (for some special die)
- Independence: Given events E_1 and E_2, they are independent if$p(E_1 \bigwedge E_2) = p(E_1)p(E_2)$(the probability that both events appear is equal to the product of the probability that one event appears) $p(E_1|E_2) = \frac{p(E_1 \bigwedge E_2)}{p(E_2)} = p(E_1)$ (the conditional probability of E_1 given the event E_2 is equal to the probability of E_1)
- example of independence: The die roll 2 and 3 are independent; Events E_2 and E_4 are also independent: $E_2 \bigwedge E_4 = \{1, 3, 5\} \bigwedge \{1, 2\} = \{1\}$ $p(E_2 \bigwedge E_4) = p(\{1\}) = \frac{1}{6}$ $p(E_2)p(E_4) = p(\{1, 3, 5\})p(\{1, 2\}) = \frac{1}{2}\frac{1}{3} = \frac{1}{6}$

3.3.2 Independent Copies

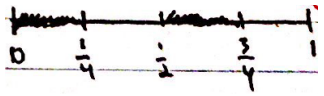
- Coin Flip: 2 independent copies: $C_1 = \{H, T\}, C_2 = \{H, T\} \Rightarrow$ $C_1 \times C_2$

	H^{P_H}	T^{P_T}
$P_H H$	HH	HT
$P_T T$	TH	TT

  - Event: $\{HT, HT\} \Rightarrow$ exactly one head $\{HT, HH, TH\} \Rightarrow$at least one head
  - Distribution: $p((H, T)) = p_1(H)p_2(T)$
  - Generalized: K-coins **independent**. $\{H, T\}^K = \{K - tuples\ of\ H, T\}$ Distribution. $p((H, ..., T)) = p_1(H)...p_k(T)$ Can generalize this to infinite protucts as well. (Doesn't hold for dependent events)

3.3.3 Continuous Distributions

- Ex: Uniform Distribution on $[0, 1]$. Outcomes: $[0, 1]$ Events: Ex: the event $[0, 0.5]$ means random number ≤ 0.5 Ex: the event $[0, 0.25] \bigcup [0.5, 0.75]$ means random number less than 0.025 or between 0.5 and 0.75

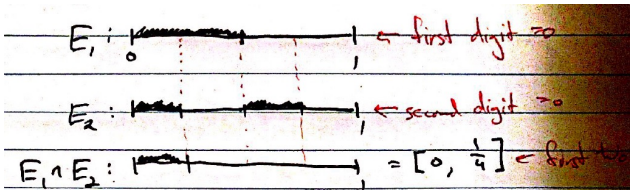


- Distribution: A distribution a rule which gives the probability of any event. Properties to satisfy: if events $E_1, E_2, ..., E_n, ...$satisfy that $E_i \bigcap E_j = \varnothing$, then
  1. $p(E_1 \bigcup ... \bigcup E_n \bigcup ...) = \sum_{i=1}^{\infty} p(E_j)$
  2. $p(E) = \int_E d_x$, "length of E"
  3. examples 1: $E_1 = [0, 0.5]$

$$P(E_1) = \int_{E_1} d_x = \int_0^{1/2} dx = \frac{1}{2}.$$

4. examples 2: $E_2 = [0, 0.25] \bigcup [0.5, 0.75]$

$$p(E_2) = \int_{E_2} dx = \int_0^{1/4} dx + \int_{1/2}^{3/4} dx = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$



- Independence: Two events E_1, E_2 are independent if

$$p(E_1 \cap E_2) = p(E_1)\, p(E_2)$$

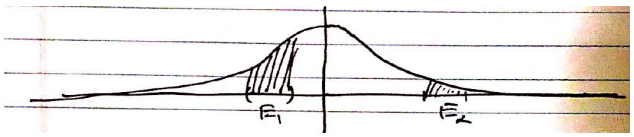
For example, $E_1 \cap E_2 = [0, 0.5] \cap \left(\left[0, \frac{1}{4}\right] \cup \left[\frac{1}{2}, \frac{3}{4}\right]\right)$

$$p\left(E_1 \bigcap E_2\right) = \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} = p(E_1)\, p(E_2)$$

Frist two binary digits are like two independent coin flips. Every binary digit is like an independent coin flip, so we ca think of the random number as being an infinite sequence of coin flips. In general, we'll consider distribution defined by a probability density function p(x). The probability of an event is given by $P(E) = \int_{E_1} p(x) d_x$

– Outcomes: $[0, 1]$
– Density function: $p(x) = 1$ ($\int_0^1 p(x) = 1$)

3.3.4 Gaussian/ Normal Distribution



- Outcomes:
- Events: Subsets of R
- Density function: $P(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ Means that for any event E,

$$p(E) = \int_E \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx.$$

- Cummulative distribution Function

$$F(x) = \int_{-\infty}^{x} p(t)dt = p(t < x)$$

Note that $\lim_{x\to\infty} F(x) = 1$.

3.4 Random, Variable, Mean, Variance

- Recall:
  - Set if outcomes: Ω
  - Event: Subset of outcomes: E
  - Probability Distribution: $p(x)$

3.4.1 Random Variable

- Defin: A random variable X is a function $X : \Omega \to S$ Here $S = R, R^d$, but could be arbitrary.
- Ex: Rolling a die:
  - Outcomes: $\Omega = 1, 2, ..., 6$
  - Events are subsets of Ω
  - Distribution: $p(1) = p(2) = ... = p(6) = \frac{1}{6}$ Suppose we roll the die, then if the die comes up d times, you win d dollars, minus 1 dollar if it's even. The amount you win is a random variable.

$$\begin{aligned} &X_1 : \Omega \to R \\ &X_1(1) = 1 \\ &X_1(2) = 1 \\ &X_1(3) = 3 \\ &X_1(4) = 3 \\ &X_1(5) = 5 \\ &X_1(6) = 5 \end{aligned}$$

  Can define multiple random variables on a single outcome space Ω.
- Ex: Rolling a die. If the die comes up d times, you get d dollars. If d is even, then you give a dollar to your friend. $X_1 \leftarrow$ your winnings, $X_2 \leftarrow$ your friends winnings.

$$\begin{aligned} X_2 : \Omega \to \mathbb{R}, &X_2(1) = 0, \quad X_2(2) = 1 \\ &X_2(3) = 0, \quad X_2(4) = 1 \\ &X_2(5) = 0, \quad X_2(6) = 1 \end{aligned}$$

From here on out Ω will be fixed, we talk about different random variables on Ω.

3.4.2 Mean of random variable

- Defin: Mean of a random variable X. Expectation of X:

$$\mathbb{E}[X] = \sum_{\omega\in\Omega} p(\omega)X(\omega) \left(= \int_{\Omega} X(\omega)p(\omega)d\omega \right)$$

- Ex:

$$\mathbb{E}[X_1] = \frac{1}{6}(1+1+3+3+5+5) = 3$$
$$\mathbb{E}[X_2] = \frac{1}{6}(0+1+0+1+0+1) = \frac{1}{2}$$

3.4.3 Variance of Random Variables

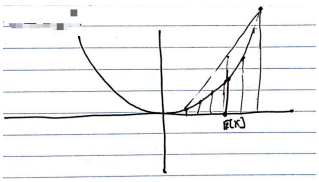
- Defin: Variance of a Random Variable X:

$$V[X] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$

- Ex:

$$\begin{aligned} &V[X_1] = \mathbb{E}\left[X_1^2\right] - (3)^2 \\ &= \tfrac{1}{6}\left(1^2 + 1^2 + 3^2 + 3^2 + 5^2 + 5^2\right) - 9 \\ &= \tfrac{1}{3}(1 + 9 + 25) - 9 \\ &= \tfrac{35}{3} - 9 = \tfrac{8}{3} \end{aligned}$$

Variance measures "how much X deviates from it's average".



3.4.4 Independenve of Random variables

- Defin: Two random variables X_1, X_2 are independent if for any α, β, $E_1 = \{w : X_1(\omega) < \alpha\}, E_2 = \{w : X_2(\omega) < \beta\}$ are independent events.
- Ex: X_1, X_2 are independent: if $(\alpha, \beta) \quad \alpha = 4, \quad \beta = \frac{1}{2}$,

$$E_1 = \{w : X_1(u) < 4\} = \{1, 2, 3, 4\}$$
$$E_2 = \left\{w : X_2(\omega) < \tfrac{1}{2}\right\} = \{1, 3, 5\}$$

$$p(E_1 \bigcap E_2) = p(E_1)\, p(E_2)$$
$$p(\{1, 3\}) = p(E_1)\, p(E_2)$$

3.4.5 Properties of E, V, Independence

– If X_1, X_2 are random variable, then

$$(a_1 X_1 + a_2 x_2)(w) = a_1 x_1(w) + a_2 x_2(w)$$

$$\mathbb{E}[a_1 X_1 + a_2 X_2] = a_1 \mathbb{E}[X_1] + a_2 \mathbb{E}[X_2]$$

– If X_1, X_2 are independent random variables, then

$$\mathbb{E}[X, X_2] = \mathbb{E}[X_1]\,\mathbb{E}[X_2]$$

– From this, we get if X_1, X_2 are independent,

$$V[X_1 + X_2] = V[X_1] + V[X_2]$$

$$V[a_1 x_1 + a_2 x_2] = a_1^2 V[x_1] + a_2^2 V[x_2]$$

3.5 Probability interpretation of logistic regression

Logistic regression models the probabilities for classification problems with two possible outcomes. It's an extension of the linear regression model for classification problems.

- Input: d-dimension feature vector $x \in R^d$
- Output: a class or label $l \in \{1, \ldots, k\}$ (how to classify)
- Example 1: Image classfication: Given input image $x \in R^{n \times n}$, predict a label for the image. e.g. cat/ dog; MNIST: $\{0, \ldots, 9\}$; CIFAR-10, $\{0, \ldots, 9\}$
- Example 2: Binary Classfication: Given some medical data $x \in R^d$(features, like heart pressure, resting heart rate, family history, etc), we try to predict incidence of heart disease. The label will be binary $\{0, 1\}$, called binary classification.

3.5.1 Logistic Regression Model

Given input feature vector $x \in R^n$, the model predicts a probability distribution over the labels $\{0, \ldots, k\}$.

$$\text{affine linear map: } Ax + b: \quad R^n \to R^k$$

with $A \in R^{k \times n}$, $x \in R^n$, $b \in R^k$.

$$\text{softmax}(Ax + b): \quad R^n \to R^k \xrightarrow{softmax} P(\{1, \ldots, k\})$$

Softmax:

1. Input: $y \in \mathbb{R}^k = \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix}$
2. Output: Distribution $p(j) = \frac{e^{y_j}}{\sum_{i=1}^{n} e^{y_i}}$
3. Take exponential and normalize

Logistic Regression Model: Given input (feature/data) $x \in R^n$, we return the distribution softmax $(Ax + b)$ where A, b are parameters.

- Example 1: Divide data into two classes: parameters are $\mathbf{a} \in \mathbb{R}^n, b \in \mathbb{R}$

$$A \in \mathbb{R}^{1\times n}, \quad Ax + b \in \mathbb{R}.$$

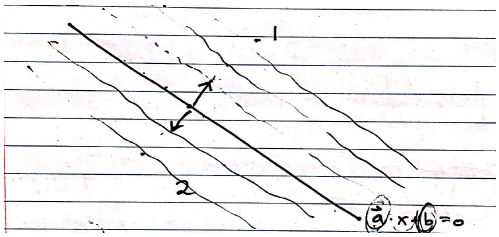
The probability that the data x belongs to class 1 is

$$P(1) = \frac{e^{\mathbf{a}\cdot x+b}}{e^{\mathbf{a}\cdot x+b}+1},$$

and the probability that the data x belongs to class 2 is

$$P(2) = \frac{1}{e^{\mathbf{a}\cdot x+b}+1}.$$

If $\mathbf{a}\cdot x + b = 0$, $P(1) = P(2) = \frac{1}{2}$. We don't know how to classify the data lying on the line $\mathbf{a}\cdot x + b = 0$ as shown in the figure below. Note that



$$\frac{p(1)}{p(2)} = e^{\mathbf{a}\cdot x+b}, \quad \log\left(\frac{p(1)}{p(2)}\right) = \mathbf{a}\cdot x + b.$$

By the above equation, $\mathbf{a}$ means which feature is important. Logarithm of the odds: $\log\left(\frac{p(1)}{p(2)}\right)$. Assumption: $\log\left(\frac{p(1)}{p(2)}\right)$ is linear in the feature vector

3.5.2 Learning the parameters a, *b* from data

Data: feature vectors x and corresponding labels l. Given data

$$\{(x_1, l_1), \ldots, (x_n, l_1)\} = D$$

How can we estimate A, b?

3.6 Maximamum Likelihood

If parameters A and b are known, for any data $x_1 \in \mathbb{R}^n$, model gives softmax $(Ax_1 + b)$

$$\frac{1}{\sum_{i=1}^{k} e^{\mathbf{a_i}\cdot x_i+b_i}}\begin{pmatrix} e^{\mathbf{a}_1\cdot x_1+b_1} \\ \vdots \\ e^{a_k\cdot x_1+b_k}\end{pmatrix} = \begin{pmatrix} p(1) \\ \vdots \\ p(k)\end{pmatrix}$$

This means that the probability that the model assigns to l_1 is

$$p(l) = \frac{1}{\sum_{i=1}^{k} e^{a_i\cdot x_i+b_i}}\cdot e^{\mathbf{a}_l\cdot x_1+b_l}$$

Instead of considering this probability, we consider its negtive logarithm:

$$-\log p(l) = \log\left(\sum_{i=1}^{k} e^{a_i\cdot x_i+b_i}\right) - (\mathbf{a}_l\cdot x_1 + b_l)$$

Since data points are independent, so we take product of the probability which leads to the logistic regression loss. Logistic Regression Loss:

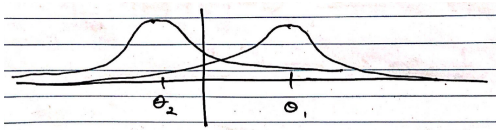
$$L_D(A,b) = \sum_{(x,l)\in D}\left[\log\left(\sum_{i=1}^{k} a_i\cdot x + b_i\right) - (\mathbf{a}_l\cdot x + b_l)\right]$$

We want to maximize the probability that the model assigns to l, that means we need to minimize $-\log p(l)$. So we need to find

$$(A,b) = \min_{A,b} L_D(A,b).$$

3.7 Basic Statistical Learning Theory

- Goal: Estimate an unknown probability distribution D on a set X from samples (i,i,d) $x_1,\ldots,x_n \in X$
- Introduce a family of distributions P_θ for $\theta \in \Theta$ and try to choose θ to "match" the samples.
  - Maximum Likelihood Estimate: Choose θ to maximize the probability of the samples.
  - Example: Let $X = R$, have some samples $x_1,\ldots,x_n$ drawn from a distribution D, say P_θ is a Gaussian with variance 1, centered at $\theta \in \mathbb{R} = \Theta$, i.e. density $p_\theta(x) = \frac{1}{\sqrt{2\pi}}e^{-(x-\theta)^2/2}$



  - Use the samples to find the center θ.

3.7.1 Maximum Likelihood Estimate(MLE)

- Given $\theta \in \Theta (= \mathbb{R}$ for this example), what is the probability of the data $\left\{x_j\right\}_{j=1}^n$?
  - Samles independent: Likelihood function(as a function of σ)

$$P_\theta\left(\left\{x_j\right\}_{j=1}^n\right) = \prod_{j=1}^n p_\theta\left(x_j\right) = \frac{1}{(\sqrt{2\pi})^n} \prod_{j=1}^n e^{-\left(x_j-\theta\right)^2/2} = \frac{1}{(2\pi)^{n/2}} e^{-\sum_{j=1}^n \left(x_j-\theta\right)^2/2}$$

  - MLE: Choose θ to maximize this!
  - Often it's useful to consider log likelihood function $\log\left(P_\theta\left(\left\{x_j\right\}_{j=1}^n\right)\right)$

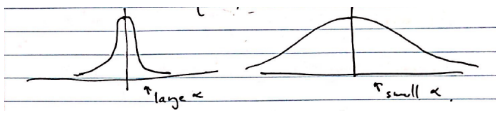
$$\theta^* = \text{argmax}_{\theta\in\Theta} \log\left(P_\theta\left(\left\{x_j\right\}_{j=1}^n\right)\right)$$
$$\left(\text{argmin}_{\theta\in\Theta} - \log\left(P_\theta\left(\left\{x_j\right\}_{j=1}^n\right)\right)\right)$$

- For this example:

$$\log\left(P_\theta\left(\left\{x_j\right\}_{j=1}^n\right)\right) = -\log(2\pi)\cdot\left(\frac{n}{2}\right) - \sum_{j=1}^n \frac{\left(x_j-\theta\right)^2}{2}$$

$$\theta^* = \text{argmin}_{\theta\in\mathbb{R}} \sum_{j=1}^n \frac{\left(x_j-\theta\right)^2}{2}.$$

$$\theta^* = \frac{1}{n}\sum_{j=1}^n x_j.$$



3.8 Classfication/ Logistic Regression

- For Classification: X is feature space and Y is label space

$$\widetilde{X} = X \times Y$$

- We have samples $\left\{\left(x_j, y_j\right)\right\}_{j=1}^n$
- Suppose that D is an unknown distribution on $\widetilde{X}$, but we're only trying to estimate $D_{Y|X}$ as a function of X. (Given the feature X, what is the possible label.)

- Introduce parameters $\theta \in \Theta$, we define $p(y|x,\theta)$ (called the model)
- Now choose θ to match the data $\left\{\left(x_j, y_j\right)\right\}_{j=1}^n$
- MLE:

$$\begin{aligned}\theta^* &= \text{argmax}_{\theta\in\Theta}\, p\left(\{y_i\}_{j=1}^n \,|\, \left\{x_j\right\}_{j=1}^n, \theta\right)\\ &= \text{argmax}_{\theta\in\Theta} \prod_{j=1}^n p\left(y_j|x_j,\theta\right)\\ &= \text{argmin}_{\theta\in\Theta} \sum_{j=1}^n -\log\left(p\left(y_j|x_j,\theta\right)\right)\end{aligned}$$

- Example: Logistic Regression:
  - Model
    $$p(y|x,\theta) = p(y|x,w,b)$$
    with $W \in \mathbb{R}^{d\times d}$ and $b \in \mathbb{R}^k$ (parameters for affine linear map)
  - d: dimension of features, i.e. number of pixels
  - k: number of classes

$$p(y|x,w,b) = \text{ softmax}\,(Wx+b) = \frac{1}{\sum_{i=1}^k e^{w_i\cdot x+b_i}}\begin{pmatrix} e^{w_1\cdot x+b_1}\\ \vdots \\ e^{w_k\cdot x+b_k}\end{pmatrix}$$

- Need to calculate:

$$\begin{aligned}&-\log\left(p\left(y_i|x_i,w,b\right)\right)\\ &= -\log\left(\frac{1}{e^{wx_i+b}\cdot\mathbb{I}}e^{wx_i+b}\cdot y_i\right)\\ &= \log\left(e^{wx_i+b}\cdot 1\right) - \log\left(e^{wx_i+b}\cdot y_i\right)\end{aligned} \tag{3.1}$$

  with $y_i = (0,\cdots,0,1,0,\cdots,0)$(only the i-th entry is 1, all others are 0) and all the entries of $\mathbb{I}$ are 1.

$$(w,b)^* = \underset{w,b}{\text{argmin}} \sum_{i=1}^n \log\left(e^{wx_i+b}\cdot\mathbb{I}\right) - \log\left(e^{wx_i+b}\cdot y_i\right)$$

3.9 Bayesian Approach to Machine Learning

3.9.1 Goal

- Goal: Estimate an unknown distribution on X from data $\left\{x_j\right\}_{j=1}^n$
  - Build a model
    - Set of parameters Θ
    - Family of distribution on X,

$$p(x|\theta) \quad \text{with} \quad \theta\in\Theta$$

· Prior distribution on the parameters Θ,

$$q(\theta).$$

– Use Bayes' Law

$$p(\theta|x)p(x) = p(x \text{ and } \theta) = p(x|\theta)p(\theta)$$

– Recall if A_1, A_2 are events:

$$p(A_1|A_2) = \frac{P(A_1 \cap A_2)}{P(A_2)}$$

$$p(\theta|x) = \frac{p(x|\theta)q(\theta)}{p(x)}$$

$$p(\theta|x) \sim p(x|\theta)q(\theta)$$

where $p(\theta|x)$ is the posteriori distribution, $q(\theta)$ is the prior distribution and $p(x|\theta)$ is the likelihood function.

- Stant with: prior $q(\theta)$
- Add data x, replace q with the posterior

$$p(\theta|x) \sim p(x|\theta)q(\theta)$$

- More data: multiply by likelihood function and then normalize
- Left with a posterior distribution
  – Sample from posterion distribution to approximate

$$P_{pred}(x) = \int_{\Theta} p(x|\theta)p(\theta)d\theta$$

– Choose θ to maximize posterior distribution

$$\begin{aligned}\theta^* &= \arg\max_{\theta\in\Theta} p(\theta|x) \\ &= \arg\min_{\theta\in\Theta} -\log p(\theta|x) \\ &= \arg\min_{\theta\in\Theta} -\log(p(x|\theta)) - \log(q(\theta)) + \log(p(x)) \\ &= \arg\min_{\theta\in\Theta} -\log(p(x|\theta)) - \log(q(\theta))\end{aligned}$$

where $-\log(p(x|\theta))$ is the negative log likelihood and $-\log(q(\theta))$ is the regularization coming from prior.

3.9.2 Example: Image Classification/ Logistic Regression

- Images $x \in X = \mathbb{R}^d$
- Labels $y \in Y = \{e_1, \ldots, e_k\}$ with k is the dimension of labels
- Data $\left\{(x_j, y_j)\right\}_{j=1}^n$

- Model $\theta = (W, b) \in \mathbb{R}^{k\times d} \times \mathbb{R}^k$. By (3.1),

$$p(y|x, \theta) = \frac{e^{Wx+b} \cdot y}{e^{Wx+b} \cdot \mathbb{I}}$$

- Prior distribution: suppose it is a Gaussian,

$$q(W, b) = Ce^{-\alpha\left(\|W\|_2^2+\|b\|_2^2\right)}$$

  – Calculate the posterior: here $q(W, b) = p(W, b|x)$,

$$p(W, b|x, y) = \frac{p(y|x, W, b)p(W, b|x)}{p(y|x)}$$

$$p(y|x) = \int_\Theta p(y|x, W, b)q(W, b)d\theta$$

$$\begin{aligned}
(W, b)^* &= \arg\min_{W,b} -\log\left(p\left(W, b|\left\{x_j, y_j\right\}_{j=1}^n\right)\right) \\
&= \arg\min_{W,b} -\log\left(p\left(\left\{y_j\right\}_{j=1}^n | \left\{x_j\right\}_{j=1}^n, W, b\right)\right) - \log(q(W, b)) \\
&= \arg\min_{W,b} -\log\left(\prod_{i=1}^n p\left(y_j|x_j, W, b\right)\right) - \log(q(W, b)) \\
&= \operatorname{argmin}_{W,b} \sum_{j=1}^n \log\left(e^{Wx+b} \cdot \mathbb{I}\right) - \log\left(e^{Wx_j+b} \cdot y_j\right) + \alpha\left(\|W\|_2^2 + \|b\|_2^2\right).
\end{aligned}$$

3.10 General Covariance

Let $X \in V$ be a random variable living in an inner product space V with $E[X] = 0$ Define: The covariance of X is the quadratic form $\mathrm{Cov}(X) : V \to R$ defined by

$$\mathrm{Cov}(X)(v) = E\left[\langle X, v\rangle^2\right]$$

If V is finite dimensional and $e_1, \ldots, e_n$ is an orthonormal matrix then

$$\langle X, v\rangle = X^\top v = v^\top X$$

and so

$$\mathrm{Cov}(X)(v) = \mathbb{E}\left[v^\top XX^\top v\right] = v^\top \mathbb{E}\left[XX^\top\right] v$$

Sometimes, we may define a scalar variance

$$\mathrm{Var}(X) = \mathbb{E}\left[\|X\|^2\right] = \mathrm{tr}\left(\mathbb{E}\left[XX^\top\right]\right)$$

3.10.1 Whitening

Consider transforming the random variable X with a linear transformation matrix A. Then

$$Cov(AX) = \mathbb{E}\left[AXX^\top A^\top\right] = A\mathbb{E}\left[XX^\top\right]A^\top = AVA^T$$

where $V = \mathbb{E}\left[XX^\top\right]$ is the coveriance of X. So if we choose A s.t.

$$AVA^\top = I \Leftrightarrow V = A^{-1}A^{-T},$$

then

$$Cov(AX) = I,$$

so AX is "white noise". Clearly there are many choice for A. Namely given an A, OA also works for any orthonormal O.

3.10.2 Batch Normalization

- Calculating an appropriate A is generally competationally too expensive. The most efficient way to do it is likely a Cholesky factorization.
- In BN, we instead choose A diagonal so that

$$\text{diag}\left(A \cup A^T\right) = (1, 1, \ldots, 1)$$

- Of course the off-diagonal entries of $A \cup A^T$ will generally not be O.

3.10.3 Central Limiting Theorem

Let $X_1, X_2, \ldots, X_n$ be independent copies of $X \in \Omega$ with $E[X] = 0$. Then $V = E\left[XX^T\right]$. Consider

$$\bar{X}_n = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_i.$$

As $n \to \infty$, $\bar{X}_n$ converges to a Gaussian distribution, namely,

$$\lim_{n\to\infty} \bar{X}_n = \text{Gaussian}(0, V)$$

with distribution

$$P(X) = \frac{1}{(\sqrt{2n}\det(V))^d} e^{\frac{-x^T V^{-1} x}{2}} dx.$$

4

Training Algorithms

4.1 Optimization: gradient descent method

We finish the study of logistic regression with solving the next optimization problem

$$\min_{\boldsymbol{\theta}} L_\lambda(\boldsymbol{\theta}). \tag{4.1}$$

For simplicity, let us just consider the next general optimization problem

$$\min_{x\in\mathbb{R}^n} f(x). \tag{4.2}$$

A general approach: line search method

Here we propose the next descent scheme to produce $\{x_t\}_{t=1}^{\infty}$

$$x_{t+1} = x_t + \eta_t p_t, \tag{4.3}$$

where η_t is called the step size in optimization and also learning rate in machine learning. p_t is called the descent direction, which is the critical component of this algorithm. There is a series of optimization algorithms which follow the above form just using different choices of p_t.

The main method that we will discuss here is the so-called gradient descent method.

4.1.1 Multi-variable calculus

To begin with the gradient based optimization, it is necessary to review some multi-variable calculus aspects and definition of convex functions.

At the very beginning, let us recall the definition of gradient and Hessian matrix for function $f : \mathbb{R}^n \to \mathbb{R}$.

Definition 7. *Given objective function $f : \mathbb{R}^n \to \mathbb{R}$ and $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$, the gradient of $f(x)$ is defined by*

$$g(x) := \nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix}, \tag{4.4}$$

Then, the next natural question is what a good choice of p_t is? We have the next theorem to show why gradient direction is a good choice for p_t.

Before that, let us introduce one often-used inequality.

Lemma 16 (Cauchy-Schwarz inequality). *For any $\mathbf{p} = (p_1, \ldots, p_n)^T$ and $\mathbf{q} = (q_1, \ldots, q_n)^T$, we have*

$$\left(\sum_{i=1}^{n} p_i q_i\right)^2 \le \left(\sum_{i=1}^{n} p_i^2\right)\left(\sum_{i=1}^{n} q_i^2\right), \tag{4.5}$$

where equality holds if and only if for some $k \in \mathbb{R}$, $\frac{p_i}{q_i} = k$, or in inner form:

$$|\langle \mathbf{p}, \mathbf{q} \rangle| \le \|\mathbf{p}\| \cdot \|\mathbf{q}\|. \tag{4.6}$$

Proof. In the case of $\mathbf{q} = \mathbf{0}$, the inequality holds. Now assume $\mathbf{q} \neq \mathbf{0}$, then

$$\begin{aligned} \|\mathbf{p} - \lambda \mathbf{q}\|^2 =& \langle \mathbf{p}, \mathbf{p} \rangle - 2\lambda \langle \mathbf{p}, \mathbf{q} \rangle + \langle \lambda \mathbf{q}, \lambda \mathbf{q} \rangle \\ =& \|\mathbf{p}\|^2 - 2\lambda \langle \mathbf{p}, \mathbf{q} \rangle + \lambda^2 \|\mathbf{q}\|^2, \end{aligned} \tag{4.7}$$

Due to that $\|\mathbf{p} - \lambda \mathbf{q}\|^2 \ge 0$ for any $\lambda \in \mathbb{R}$ and the discriminant of root of quadratic equation,

$$\Delta(\lambda) = 4\langle \mathbf{p}, \mathbf{q} \rangle^2 - 4\|\mathbf{p}\|^2 \|\mathbf{q}\|^2 \le 0 \tag{4.8}$$

If the inequality holds as an equality, which means there exists one $k \in \mathbb{R}$ such that $\|\mathbf{p} - \lambda\mathbf{q}\|^2 = 0$, or so-called $\mathbf{p}$ and $\mathbf{q}$ are linearly dependent. Conversely, if $\mathbf{p}$ and $\mathbf{q}$ are linearly dependent, then there is only one solution $\lambda = k$ to equation $\|\mathbf{p} - \lambda\mathbf{q}\|^2 = 0$, therefore, $\Delta(\lambda) = 0$. □

Theorem 4. *To choose descent directions $p \in \mathbb{R}^n$, we have*

$$-\frac{\nabla f(x)}{\|\nabla f(x)\|} = \arg\min_{p\in\mathbb{R}^n, \|p\|=1} \left.\frac{\partial f(x+\eta p)}{\partial \eta}\right|_{\eta=0}. \tag{4.9}$$

Proof. First, we have

$$\left.\frac{\partial f(x+\eta p)}{\partial \eta}\right|_{\eta=0} = \nabla f(x) \cdot p. \tag{4.10}$$

Then notice the Cauchy-Schwarz inequality and the constrain that $\|p\| = 1$,

$$|\nabla f(x) \cdot p| \le \|\nabla f(x)\|\|p\| = \|\nabla f(x)\|, \tag{4.11}$$

which mean that

$$\nabla f(x) \cdot p \ge -\|\nabla f(x)\|, \tag{4.12}$$

and the equality holds when $p = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$. □

Corollary 1. *Locally, $f(x)$ decreases most rapidly along the negative gradient direction.*

Here is a simple diagram for this property.

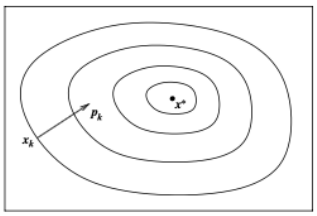


Fig. 4.1. Negative Gradient Direction: x_k is current point, p_k is the negative gradient of x_k, i.e., $-\nabla f(x_k)$

.

Since at each point, $f(x)$ decreases most rapidly along the negative gradient direction, it is then natural to choose the search direction in (4.3) in the negative gradient direction and the resulting algorithm is the so-called gradient descent method.

Algorithm 2 Gradient Descent Method

Given the initial guess x_0, learning rate $\eta_t > 0$
For t=1,2,$\cdots$,

$$x_{t+1} = x_t - \eta_t \nabla f(x_t), \tag{4.13}$$

In practice, we need a "stopping criterion" that determines when the above gradient descent method to stop. One possibility is

While $S(x_t; f) = \|\nabla f(x_t)\| \leq \epsilon$ or $t \geq T$

for some small tolerance $\epsilon > 0$ or maximal number of iterations T. In general, a good stopping criterion is hard to come by and it is subject that has called a lot of research in optimization for machine learning.

4.2 Convex functions and convergence of gradient descent

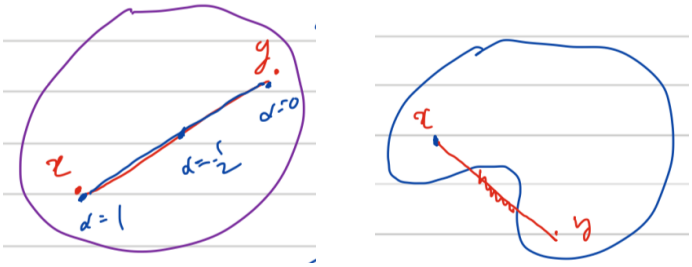
4.2.1 Convex function

Then, let us first give the definition of convex sets.

Definition 8 (Convex set). *A set C is convex, if the line segment between any two points in C lies in C, i.e., if any $x, y \in C$ and any α with $0 \le \alpha \le 1$, there holds*

$$\alpha x + (1-\alpha)y \in C. \tag{4.14}$$

Here are two diagrams for this definition about convex and non-convex sets.



Following the definition of convex set, we define convex function as following.

Definition 9 (Convex function). *Let $C \subset \mathbb{R}^n$ be a convex set and $f : C \to \mathbb{R}$:*

1. *f is called* **convex** *if for any $x, y \in C$ and $\alpha \in [0,1]$*

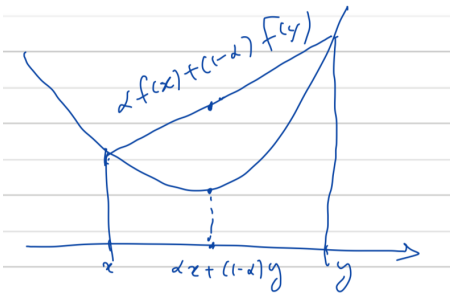
$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y). \tag{4.15}$$

2. *f is called* **strictly convex** *if for any $x \ne y \in C$ and $\alpha \in (0,1)$:*

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha) f(y). \tag{4.16}$$

3. *A function f is said to be (strictly)* **concave** *if $-f$ is (strictly) convex.*

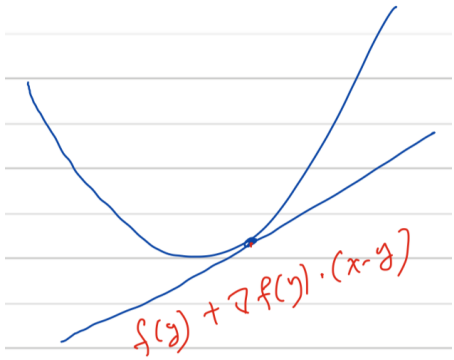
We also have the next diagram for convex function definition.



Lemma 17. *If $f(x)$ is differentiable on $\mathbb{R}^n$, then $f(x)$ is convex if and only if*

$$f(x) \geq f(y) + \nabla f(y) \cdot (x - y), \forall x, y \in \mathbb{R}^n. \tag{4.17}$$

Based on the lemma, we can first have the next new diagram for convex functions.



Proof. Let $z = \alpha x + (1 - \alpha)y, 0 \leq \alpha \leq 1, \forall x, y \in \mathbb{R}^n$, we have these next two Taylor expansion:

$$\begin{aligned} f(x) &\geq f(z) + \nabla f(z)(x - z) \\ f(y) &\geq f(z) + \nabla f(z)(y - z). \end{aligned} \tag{4.18}$$

Then we have

$$\begin{aligned} &\alpha f(x) + (1 - \alpha) f(y) \\ \geq &f(z) + \nabla f(z)[\alpha(x - z) + (1 - \alpha)(y - z)] \\ = &f(z) \\ = &f(\alpha x + (1 - \alpha)y). \end{aligned} \tag{4.19}$$

Thus we have

$$\alpha f(x) + (1 - \alpha) f(y) \geq f(\alpha x + (1 - \alpha)y). \tag{4.20}$$

This finishes the proof.

On the other hand (**homework**): if $f(x)$ is differentiable on $\mathbb{R}^n$, then $f(x) \geq f(y) + \nabla f(y) \cdot (x - y), \forall x, y \in \mathbb{R}^n$ if $f(x)$ is convex. □

Definition 10 (λ-strongly convex). *We say that $f(x)$ is λ-strongly convex if*

$$f(x) \geq f(y) + \nabla f(y) \cdot (x - y) + \frac{\lambda}{2}\|x - y\|^2, \quad \forall x, y \in C, \tag{4.21}$$

for some $\lambda > 0$.

Example 3. Consider $f(x) = \|x\|^2$, then we have

$$\frac{\partial f}{\partial x_i} = 2x_i, \nabla f = 2x \in R^n. \tag{4.22}$$

So, we have

$$\begin{aligned} & f(x) - f(y) - \nabla f(y)(x-y) \\ =& \|x\|^2 - \|y\|^2 - 2y(x-y) \\ =& \|x\|^2 - \|y\|^2 - 2xy + 2\|y\|^2 \\ =& \|x\|^2 - 2xy + \|y\|^2 \\ =& \|x-y\|^2 \\ =& \frac{\lambda}{2}\|x-y\|^2, \quad \lambda = 2. \end{aligned} \tag{4.23}$$

Thus, $f(x) = \|x\|^2$ is 2-strongly convex

Example 4 (Homework). Actually, the loss function of the logistic regression model

$$L(\theta) = -\log P(\theta), \tag{4.24}$$

is convex as a function of θ.

Furthermore, the loss function of the regularized logistic regression model

$$L_\lambda(\theta) = -\log P(\theta) + \lambda\|\theta\|_F^2, \lambda > 0 \tag{4.25}$$

is λ'-strongly convex (λ' is related to λ) as a function of θ.

We also have these following interesting properties of convex function.

Properties 5 (basic properties of convex function) *[Homework]*

1. *If $f(x)$, $g(x)$ are both convex, then $\alpha f(x) + \beta g(x)$ is also convex, if $\alpha, \beta \geq 0$.*
2. *Linear function is both convex and concave. Here, $f(x)$ is concave if and only if $-f(x)$ is convex.*
3. *If $f(x)$ is a convex convex function on $\mathbb{R}^n$, then $g(y) = f(Ay + b)$ is a convex function on $\mathbb{R}^m$. Here $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.*
4. *If $g(x)$ is a convex function on $\mathbb{R}^n$, and the function $f(u)$ is convex function on $\mathbb{R}$ and non-decreasing, then the composite function $f \circ g(x) = f(g(x))$ is convex.*

Proof. **Homework**: prove them by definition. □

4.2.2 On the Convergence of GD

For the next optimization problem

$$\min_{x\in\mathbb{R}^n} f(x). \tag{4.26}$$

We assume that $f(x)$ is convex. Then we say that x^* is a minimizer if $f(x^*) = \min_{x\in\mathbb{R}^n} f(x)$.

Let recall that, for minimizer x^* we have

$$\nabla f(x^*) = 0. \tag{4.27}$$

Then we have the next tw properties of minimizer for convex functions:

1. If $f(x) \geq c_0$, for some $c_0 \in \mathbb{R}$, then we have
$$\arg\min f \neq \emptyset. \tag{4.28}$$
2. If $f(x)$ is λ-strongly convex, then $f(x)$ has a unique minimizer, namely, there exists a unique $x^* \in \mathbb{R}^n$ such that
$$f(x^*) = \min_{x\in\mathbb{R}^n} f(x). \tag{4.29}$$

To investigate the convergence of gradient descent method, let recall the gradient descent method:

Algorithm 3 FGD

For: $t = 1, 2, \cdots$

$$x_{t+1} = x_t - \eta_t \nabla f(x_t), \tag{4.30}$$

where η_t is the stepsize / learning rate.

Assumption 4.31 *We make the following assumptions*

1. *$f(x)$ is λ-strongly convex for some $\lambda > 0$. Recall the definition, we have*
$$f(x) \geq f(y) + \nabla f(y)\cdot(x-y) + \frac{\lambda}{2}\|x-y\|^2,$$
then note $x^* = \arg\min f(x)$. Then we have
   - *Take $y = x^*$, this leads to*
   $$f(x) \geq f(x^*) + \frac{\lambda}{2}\|x-y\|^2.$$
   - *Take $x = x^*$, this leads to*
   $$0 \geq f(x^*) - f(y) \geq \nabla f(y)\cdot(x^*-y) + \frac{\lambda}{2}\|x^*-y\|^2,$$
   which means that
   $$\nabla f(x)\cdot(x-x^*) \geq \frac{\lambda}{2}\|x-x^*\|^2. \tag{4.32}$$

2. ∇f *is Lipschitz for some* $L > 0$*, i.e.,*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \forall x, y. \tag{4.33}$$

Thus, we have the next theorem about the convergence of gradient descent method.

Theorem 6. *For Algorithm 3, if* $f(x)$ *is* λ*-strongly convex and* ∇f *is Lipschitz for some* $L > 0$*, then*

$$\|x_t - x^*\|^2 \leq \alpha^t \|x_0 - x^*\|^2 \tag{4.34}$$

if $0 < \eta_t \leq \eta_0 = \frac{\lambda}{2L^2}$ *and* $\alpha = 1 - \frac{\lambda^2}{4L^2} < 1$.

Proof. If we minus any $x \in \mathbb{R}^n$, we can only get:

$$x_{t+1} - x = x_t - \eta_t \nabla f(x_t) - x. \tag{4.35}$$

If we take L^2 norm for both side, we get:

$$\|x_{t+1} - x\|^2 = \|x_t - \eta_t \nabla f(x_t) - x\|^2. \tag{4.36}$$

So we have the following inequality and take $x = x^*$:

(4.37)

$$\begin{aligned}
\|x_{t+1} - x^*\|^2 &= \|x_t - \eta_t \nabla f(x_t) - x^*\|^2 \\
&= \|x_t - x^*\|^2 - 2\eta_t \nabla f(x_t)^\top (x_t - x^*) + \eta_t^2 \|\nabla f(x_t) - \nabla f(x^*)\|^2 \\
&\leq \|x_t - x^*\|^2 - \eta_t \lambda \|x_t - x^*\|^2 + \eta_t^2 L^2 \|x_t - x^*\|^2 \quad (\lambda - \text{strongly convex and Lipschitz}) \\
&\leq (1 - \eta_t \lambda + \eta_t^2 L^2) \|x_t - x^*\|.
\end{aligned}$$

So, if $\eta_t \leq \frac{\lambda}{2L^2}$, then $\alpha = (1 - \eta_t \lambda + \eta_t^2 L^2) \leq 1 - \frac{\lambda^2}{4L^2} < 1$, which finishes the proof. □

Some issues on GD:

- $\nabla f(x_t)$ is very expensive to compete.
- GD does not yield generalization accuracy.

The stochastic gradient descent (SGD) method which we will discuss in the next section will focus on these two issues.

4.3 Stochastic gradient descent method and convergence theory

The next optimization problem is the most common case in machine learning.

Problem 1.

$$\min_{x\in\mathbb{R}^n} f(x), \tag{4.38}$$

where

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x). \tag{4.39}$$

One version of stochastic gradient descent (SGD) algorithm is:

Algorithm 4 SGD

Input: initialization x_0, learning rate η_t.
For: t = 0,1,2,...
Randomly pick $i_t \in \{1, 2, \cdots, N\}$ independently with probability $\frac{1}{N}$

$$x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t). \tag{4.40}$$

4.3.1 Convergence of SGD

Theorem 7. *Assume that each $f_i(x)$ is λ-strongly convex and $\|\nabla f_i(x)\| \le M$ for some $M > 0$. If we take $\eta_t = \frac{a}{\lambda(t+1)}$ with sufficiently large a such that*

$$\|x_0 - x^*\|^2 \le \frac{a^2 M^2}{(a-1)\lambda^2} \tag{4.41}$$

then

$$\mathbb{E}e_t^2 \le \frac{a^2 M^2}{(a-1)\lambda^2(t+1)}, \quad t \ge 1, \tag{4.42}$$

where $e_t = \|x_t - x^*\|$.

Proof. The L^2 error of SGD can be written as

$$\begin{aligned}
\mathbb{E}\|x_{t+1} - x^*\|^2 &\le \mathbb{E}\|x_t - \eta_t \nabla f_{i_t}(x_t) - x^*\|^2 \\
&\le \mathbb{E}\|x_t - x^*\|^2 - 2\eta_t \mathbb{E}(\nabla f_{i_t}(x_t)\cdot(x_t - x^*)) + \eta_t^2 \mathbb{E}\|\nabla f_{i_t}(x_t)\|^2 \\
&\le \mathbb{E}\|x_t - x^*\|^2 - 2\eta_t \mathbb{E}(\nabla f(x_t)\cdot(x_t - x^*)) + \eta_t^2 M^2 \\
&\le \mathbb{E}\|x_t - x^*\|^2 - \eta_t\lambda \mathbb{E}\|x_t - x^*\|^2 + \eta_t^2 M^2 \\
&= (1 - \eta_t\lambda)\mathbb{E}\|x_t - x^*\|^2 + \eta_t^2 M^2
\end{aligned} \tag{4.43}$$

The third line comes from the fact that

$$\begin{aligned}\mathbb{E}(\nabla f_{i_t}(x_t)\cdot(x_t-x^*)) &= \mathbb{E}_{i_1 i_2\cdots i_t}(\nabla f_{i_t}(x_t)\cdot(x_t-x^*))\\ &= \mathbb{E}_{i_1 i_2\cdots i_{t-1}}\frac{1}{N}\sum_{i=1}^{N}\nabla f_i(x_t)\cdot(x_t-x^*)\\ &= \mathbb{E}_{i_1 i_2\cdots i_{t-1}}\nabla f(x_t)\cdot(x_t-x^*)\\ &= \mathbb{E}\nabla f(x_t)\cdot(x_t-x^*),\end{aligned} \tag{4.44}$$

and

$$\mathbb{E}\|\nabla f_{i_t}(x_t)\|^2 \le \mathbb{E}M^2 = M^2. \tag{4.45}$$

Note when $t = 0$, we have

$$\mathbb{E}e_0^2 = \|x_0 - x^*\|^2 \le \frac{a^2M^2}{(a-1)\lambda}, \tag{4.46}$$

based on the assumption.

In the case of SDG, by the inductive hypothesis,

$$\begin{aligned}\mathbb{E}e_{t+1}^2 &\le (1-\eta_t\lambda)\mathbb{E}e_t^2 + \eta_t^2M^2\\ &\le (1-\frac{a}{t+1})\frac{a^2M^2}{(a-1)\lambda^2(t+1)} + \frac{a^2M^2}{\lambda^2(t+1)^2}\\ &\le \frac{a^2M^2}{(a-1)\lambda^2}\frac{1}{(t+1)^2}(t+1-a+a-1)\\ &= \frac{a^2M^2}{(a-1)\lambda^2}\frac{t}{(t+1)^2}\\ &\le \frac{a^2M^2}{(a-1)\lambda^2(t+2)}. \quad \left(\frac{t}{(t+1)^2}\le\frac{1}{t+2}\right)\end{aligned} \tag{4.47}$$

□

4.3.2 SGD with mini-batch

Firstly, we will introduce a natural extended version of the SGD discussed above with introducing mini-bath.

Algorithm 5 SGD with mini-batch

Input: initialization x_0, learning rate η_t.
For: t = 0,1,2,...

$$\text{Randomly pick } B_t \subset \{1, 2, \cdots, N\} \text{ independently with probability } \frac{1}{\binom{N}{m}} \text{ and } \#B_t = m.$$

$$x_{t+1} = x_t - \eta_t g_t(x_t). \tag{4.48}$$

where

$$g_t(x_t) = \frac{1}{m} \sum_{i \in B_t} \nabla f_i(x_t)$$

Now we introduce the SGD algorithm with mini-batch without replacement which is the most commonly used version of SGD in machine learning.

Algorithm 6 Shuffle SGD with mini-batch

Input: learning rate η_k, mini-batch size m, parameter initialization x_0 and denote $M = \lceil \frac{N}{m} \rceil$.
For Epoch $k = 1, 2, \ldots$

$$\text{Randomly pick } B_t \subset \{1, 2, \cdots, N\} \text{ without replacement with } \#B_t = m \text{ for } t = 1, 2, \cdots, M.$$

For mini-batch $t = 1 : M$
Compute the gradient on B_t:
Update x:

$$x \leftarrow x - \eta_k g_t(x),$$

where

$$g_t(x) = \frac{1}{m} \sum_{i \in B_t} \nabla f_i(x)$$

EndFor
EndFor

To " randomly pick $B_i \subset \{1, 2, \cdots, N\}$ without replacement with $\#B_i = m$ for $i = 1, 2, \cdots, t$", we usually just randomly shuffle the index set first and then consecutively pick every m elements in the shuffled index set. That is the reason why we would like to call the algorithm as shuffled SGD while this is the mostly used version of SGD in machine learning.

Remark 3. Let recall a general machine learning loss function

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(h(X_i;\theta), Y_i), \tag{4.49}$$

where $\{(X_i, Y_i)\}_{i=1}^N$ correspond to these data pairs. For example, $\ell(\cdot,\cdot)$ takes cross-entropy and $h(x;\theta) = \boldsymbol{p}(x;\theta)$ as we discussed in Logistic regression section.

Thus, we have the following corresponding relation

$$f(x) \leftrightsquigarrow L(\theta)$$
$$f_i(x) \leftrightsquigarrow \ell(h(X_i;\theta), Y_i).$$

5

Polynomials and Weierstrass theorem

5.1 Nonlinear classifiable sets

In the section, we will extend the linearly separable sets to nonlinear case. A natural extension is like what kernel method does in SVM for binary case, we will introduce the so-called feature mapping.

Thus, we have the following natural extension for linearly separable by using feature mapping and original definition of linearly separable.

Definition 11 (nonlinearly separable sets). *These data sets $A_1, A_2, \cdots, A_k \subset \mathbb{R}^d$ are called nonlinearly separable, if there exist a feature space $\mathbb{R}^{\tilde{d}}$ and a smooth (if it has derivatives of all orders) feature mapping*

$$\varphi : \mathbb{R}^d \mapsto \mathbb{R}^{\tilde{d}} \tag{5.1}$$

such that

$$\tilde{A}_i := \varphi(A_i) = \{\tilde{x} \mid \tilde{x} = \varphi(x), x \in A_i\}, \quad i = 1, 2, \ldots, k, \tag{5.2}$$

are linearly separable.

Remark 4. 1. This definition is also consistent with the definition of linearly separable as we can just take $\tilde{d} = d$ and $\varphi = \mathrm{id}$ if $A_1, A_2, \cdots, A_k$ are already linearly separable.
2. The kernel method in SVM is mainly based on this idea for binary case (k=2) where they use kernel functions to approximate this $\varphi(x)$.
3. For most commonly used deep learning models, they are all associated with a softmax mapping which means that we can interpret these deep learning models as the approximation for feature mapping φ.

However, softmax is not so crucial for this definition actually as we have the next equivalent result.

Theorem 8. *$A_1, A_2, \cdots, A_k \subset \mathbb{R}^d$ are nonlinearly separable is equivalent that there there exist a smooth classification function*

$$\psi : \mathbb{R}^d \mapsto \mathbb{R}^k \tag{5.3}$$

such that for all $i = 1 : k$ and $j \neq i$

$$\psi_i(x) > \psi_j(x), \quad \forall x \in A_i. \tag{5.4}$$

Proof. On the one hand, it is easy to see that if $A_1, A_2, \cdots, A_k \subset \mathbb{R}^d$ are nonlinearly separable then they we can just take

$$\psi(x) = \boldsymbol{p}(\varphi(x); \theta), \tag{5.5}$$

where the $\boldsymbol{p}(y; \theta)$ is the softmax function for linearly separable sets $\varphi(A_i)$ for $i = 1, 2, \cdots, k$.

On the other hand, let assume that ψ is the smooth classification functions for $A_1, A_2, \cdots, A_k \subset \mathbb{R}^d$. We claim that, we can take $\varphi(x) = \psi(x)$ and then

$$\varphi(A_1), \varphi(A_2), \cdots, \varphi(A_k) \subset \mathbb{R}^k \quad (\tilde{d} = k), \tag{5.6}$$

will be linearly separable. Actually, if you take $\theta = (I, 0)$ in softmax mapping $\boldsymbol{p}(x; \theta)$, then the monotonicity of e^x show that for all $i = 1 : k$ and $j \neq i$

$$\boldsymbol{p}_i(\varphi(x); \theta) = \frac{e^{\psi_i(x)}}{\sum_{i=1}^k e^{\psi_i(x)}} > \frac{e^{\psi_j(x)}}{\sum_{i=1}^k e^{\psi_i(x)}} = \boldsymbol{p}_j(\varphi(x); \theta), \quad \forall x \in A_i. \tag{5.7}$$

□

Similarly to linearly separable sets, we have the next lemme for $k = 2$.

Lemma 18. *A_1 and A_2 are nonlinearly separable is equivalent that there exists a function $\varphi : \mathbb{R}^d \mapsto \mathbb{R}$ such that*

$$\varphi(x) > 0 \quad \forall x \in A_1 \quad \textit{and} \quad \varphi(x) < 0 \quad \forall x \in A_2. \tag{5.8}$$

Proof. Based the equivalence of nonlinearly separable sets, there exists $\psi_1(x)$ and $\psi_2(2)$ such that for all $i = 1 : 2$ and $j \neq i$

$$\psi_i(x) > \psi_j(x), \quad \forall x \in A_i. \tag{5.9}$$

Then, we can just take

$$\varphi(x) = \psi_1(x) - \psi_2(x). \tag{5.10}$$

On the other hand, if there exist $\varphi(x)$, then we can construct $\psi_1(x)$ and $\psi_2(2)$ as

$$\psi_1(x) = \frac{1}{2}\varphi(x) \quad \text{and} \quad \psi_2(x) = -\frac{2}{2}\varphi(x). \tag{5.11}$$

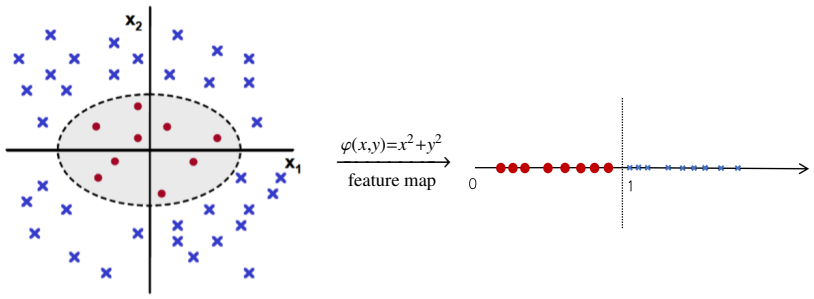
□

Remark 5. Here we mention that, we only assume that for all $i = 1 : k$ and $j \neq i$ we have $\psi_i(x) > \psi_j(x)$, $\forall x \in A_i$ for nonlinearly separable. We do not assume that $\psi_i(x) \geq 0$ or $\sum_{i=1}^k \psi_i(x) = 1$, which means that $\psi(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \\ \vdots \\ \psi_k(x) \end{pmatrix}$ is not a discrete probability distribution over all k classes.

The previous theorem shows that softmax function is not so crucial in nonlinearly separable case. Combined with deep learning models, we have the following understanding about what deep learning models are approximating.

1. If a classification model is followed with a softmax, then the it is approximating the feature mapping $\varphi : \mathbb{R}^d \mapsto \mathbb{R}^{\tilde{d}}$.
2. If the classification model dose not followed by softmax, then it is approximating $\psi : \mathbb{R}^d \mapsto \mathbb{R}^k$ directly.

Example 5. Consider $k = 2$ and $A_1 \subset \{(x, y)|x^2 + y^2 < 1\}, \quad A_2 \subset \{(x, y)|x^2 + y^2 > 1\}$, then we can have the following nonlinear feature mapping:



Here we have the following comparison for linear and nonlinear models from the viewpoint of loss functions:

Linear case (Logistic regression):

$$L_\lambda(\theta) = \sum_{j=1}^N \ell(y_j, p(x_j; \theta)) + \lambda R(\|\theta\|)$$

Nonlinear case:

$$L_\lambda(\theta) = \sum_{j=1}^N \ell(y_j, p(\varphi(x_j; \theta_1); \theta_2)) + \lambda R(\|\theta\|)$$

Remark 6. We have the following remarks.

1. $\ell(q, p) = \sum_{i=1}^k -q_i \log p_i \leftrightarrow$ cross-entropy

2. $p(x;\theta) = \text{softmax}(Wx + b)$ where $\theta = (W, b)$
3. $\theta = (\theta_1, \theta_2)$ for nonlinear case
4. $\lambda R(\|\theta\|) \leftrightarrow$ regularization term

In general, we have the following popular nonlinear models for $\varphi(x;\theta)$

1. Polynomials.
2. Piecewise polynomials (finite element method).
3. Kernel functions in SVM.
4. Deep neural networks.

In this chapter, we discuss approximation properties of spaces of polynomials and finite elements consisting of piecewise polynomials.

5.2 Approximation by polynomials and Weierstrass Theorem

Let $\alpha = (\alpha_1, \cdots, \alpha_d)$ with α_i being non-negative integers, we note $|\alpha| = \sum_{i=1}^{d} \alpha_i$ and

$$x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}.$$

We use $\mathbb{P}_m(\mathbb{R}^d)$ to define the polynomials of d-variables of degree less than m which consists functions of the form

$$\sum_{|\alpha| \le m} a_\alpha x^\alpha = \sum_{\alpha_1+\alpha_2+\ldots+\alpha_d \le m} a_{\alpha_1,\ldots,\alpha_d} x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}$$

5.2.1 Weierstrass Theorem and Proof

Important property: polynomials can approximate any reasonable function!

- dense in $C(\Omega)$ [Weierstrass theorem]
- dense in all Sobolev spaces: $L^2(\Omega)$, $W^{m,p}(\Omega)$, ...

Theorem 9. *Let $\Omega \subset R^n$ be a closed and bounded set. Given any continuous function $f(x)$ on Ω, there exists a sequence of polynomials $\{P_n(x)\}$ such that*

$$\lim_{n\to\infty} \max_{x\in\Omega} |f(x) - P_n(x)| = 0 \tag{5.12}$$

Proof. Let us first give the proof for $d = 1$ and $\Omega = [0, 1]$. Given $f : [0, 1] \to R$ be a continuous function.

Let

$$\tilde{f}(x) = f(x) - l(x) \tag{5.13}$$

where $l(x) = f(0) + x(f(1) - f(0))$. Then $\tilde{f}(0) = \tilde{f}(1) = 0$. Noting that $l(x)$ is a linear function, hence without lose of generality, we can only consider the case $f : [0, 1] \to R$ with $f(0) = f(1) = 0$.

Since f is continuous on the closed interval $[0,1]$, then f is uniformly continuous on $[0,1]$.

First we extend f to be zero outside of $[0,1]$ and obtain $f : R \to R$, then it is obviously that f is still uniformly continuous.

Next for $0 \le x \le 1$, we construct

$$p_n(x) = \int_{-1}^{1} f(x+t)Q_n(t)dt = \int_{-x}^{1-x} f(x+t)Q_n(t)dt = \int_0^1 f(t)Q_n(t-x)dt \tag{5.14}$$

where $Q_n(x) = c_n(1-x^2)^n$ and

$$\int_{-1}^{1} Q_n(x)dx = 1. \tag{5.15}$$

Thus $\{p_n(x)\}$ is a sequence of polynomials.

Since

$$\int_{-1}^{1}(1-x^2)^n dx = 2\int_0^1 (1-x^2)^n dx = 2\int_0^1 (1-x)^n(1+x)^n dx \tag{5.16}$$

$$\ge 2\int_0^1 (1-x)^n dx = \frac{2}{n+1} > \frac{1}{n}. \tag{5.17}$$

Combing with $\int_{-1}^{1} Q_n(x)dx = 1$, we obtain $c_n < n$ implying that for any $\delta > 0$

$$0 \le Q_n(x) \le n(1-\delta^2)^n \quad (\delta \le |x| \le 1), \tag{5.18}$$

so that $Q_n \to 0$ uniformly in $\delta \le |x| \le 1$ as $n \to \infty$.

Given any $\epsilon > 0$, since f in uniformly continuous, there exists $\delta > 0$ such that for any $|y-x| < \delta$, we have

$$|f(y) - f(x)| < \frac{\epsilon}{2}. \tag{5.19}$$

Finally, let $M = \max|f(x)|$, using (5.19), (5.15) and (5.18), we have

$$\big|p_n(x) - f(x)\big| = \Big|\int_{-1}^{1}(f(x+t)-f(t))Q_n(t)dt\Big| \le \int_{-1}^{1}\big|f(x+t)-f(t)\big|Q_n(t)dt \tag{5.20}$$

$$\le 2M\int_{-1}^{-\delta} Q_n(t)dt + \frac{\epsilon}{2}\int_{-\delta}^{\delta} Q_n(t)dt + 2M\int_{\delta}^{1} Q_n(t)dt \tag{5.21}$$

$$\le 4Mn(1-\delta^2)^n + \frac{\epsilon}{2} < \epsilon \tag{5.22}$$

for all large enough n, which proves the theorem.

The above proof generalize the high dimensional case easily. We consider the case that

$$\Omega = [0,1]^d.$$

By extension and using cut off function, W.L.O.G. that we assume that $f = 0$ on the boundary of Ω and we then extending this function to be zero outside of Ω.

Let us consider the special polynomial functions

$$Q_n(x) = c_n \prod_{k=1}^{d} (1 - x_k^2) \tag{5.23}$$

Similar proof can then be applied. □

5.2.2 Some issues with polynomial approximations

Curse of dimensionality

Number of coefficients for polynomials of degrees n in $\mathbb{R}^d$ is

$$N = \binom{d+n}{n} = \frac{(n+d)!}{d!n!}.$$

For example $n = 100$:

$d =$	2	4	8
$N =$	5×10^3	4.6×10^6	3.5×10^{11}

Runge's phenomenon

Consider the case where one desires to interpolate through $n+1$ equispaced points of a function $f(x)$ using the n-degree polynomial $P_n(x)$ that passes through those points. Naturally, one might expect from Weierstrass' theorem that using more points would lead to a more accurate reconstruction of $f(x)$. However, this particular set of polynomial functions $P_n(x)$ is not guaranteed to have the property of uniform convergence; the theorem only states that a set of polynomial functions exists, without providing a general method of finding one.

The $P_n(x)$ produced in this manner may in fact diverge away from $f(x)$ as n increases; this typically occurs in an oscillating pattern that magnifies near the ends of the interpolation points. This phenomenon is attributed to Runge.

Problem: Consider the Runge function

$$f(x) = \frac{1}{1 + 25x^2}$$

(a scaled version of the Witch of Agnesi). Runge found that if this function is interpolated at equidistant points x_i between -1 and 1 such that:

$$x_i = \frac{2i}{n} - 1, \quad i \in \{0, 1, \ldots, n\}$$

with a polynomial $P_n(x)$ of degree $\leq n$, the resulting interpolation oscillates toward the ends of the interval, i.e. close to -1 and 1. It can even be proven that the interpolation error increases (without bound) when the degree of the polynomial is increased:

$$\lim_{n\to\infty}\left(\max_{-1\leq x\leq 1}|f(x)-P_n(x)|\right)=+\infty.$$

This shows that high-degree polynomial interpolation at equidistant points can be troublesome.

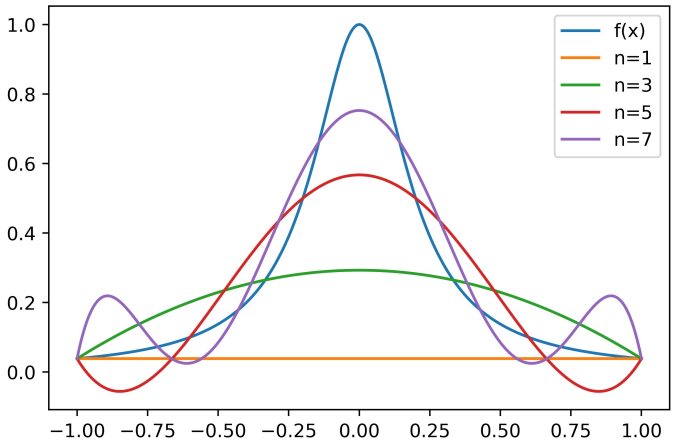


Fig. 5.1. Runge's phenomenon: Runge function $f(x)=\frac{1}{1+25x^2}$ and its polynomial interpolation $P_n(x)$.

6

Finite Element Method

6.1 Conforming linear finite element spaces

A conforming linear finite element function in a domain $\Omega \subset \mathbb{R}^d$ is a continuous function that is piecewise linear function with a grid or mesh consisting of a union of simplexes.

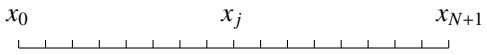


Fig. 6.1. 1D uniform grid

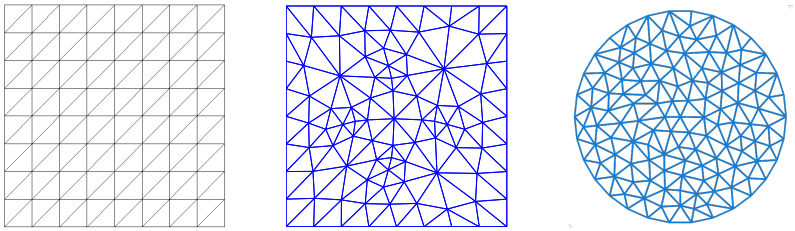


Fig. 6.2. 2D grids

6.1.1 Simplexes in $\mathbb{R}^d$

Let $\boldsymbol{x}_i = (x_{1,i}, \cdots, x_{d,i})^t, i = 1, \cdots, d+1$ be $d+1$ points in $\mathbb{R}^d$ which do not all lie in one hyper-plane. The *convex hull* of the $d+1$ points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{d+1}$ (See Figure 6.3)

$$\tau := \{\boldsymbol{x} = \sum_{i=1}^{d+1} \lambda_i \boldsymbol{x}_i \,|\, 0 \le \lambda_i \le 1, i = 1 : d+1, \sum_{i=1}^{d+1} \lambda_i = 1\} \tag{6.1}$$

is defined as a *geometric d-simplex* generated (or spanned) by the vertices $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{d+1}$. For example, a triangle is a 2-simplex and a tetrahedron is a 3-simplex. For an integer $0 \le m \le d-1$, an m-dimensional face of τ is any m-simplex generated by $m+1$ of the vertices of τ. Zero-dimenisonal faces are vertices and one-dimensional faces are called edges of τ. The $(d-1)$-face opposite to the vertex $\boldsymbol{x}_i$ will be denoted by F_i.

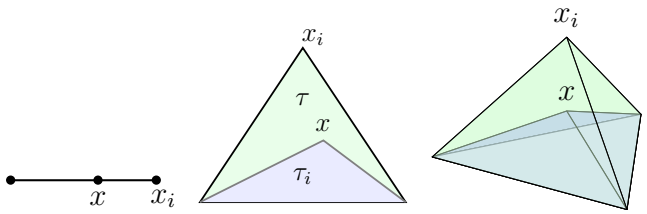


Fig. 6.3. Geometric explanation of barycentric coordinate

Barycentric coordinates

The numbers $\lambda_1(\boldsymbol{x}), \cdots, \lambda_{d+1}(\boldsymbol{x})$ are called *barycentric coordinates* of $\boldsymbol{x}$ with respect to the $d+1$ points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{d+1}$. There is a simple geometric meaning of the barycentric coordinates. Given a $\boldsymbol{x} \in \tau$, let $\tau_i(\boldsymbol{x})$ be the simplex with vertices $\boldsymbol{x}_i$ replaced by $\boldsymbol{x}$. Then it can be easily shown that

$$\lambda_i(\boldsymbol{x}) = |\tau_i(\boldsymbol{x})|/|\tau|, \tag{6.2}$$

where $|\cdot|$ is the Lebesgure measure in $\mathbb{R}^d$, namely area in two dimensions and volume in three dimensions. Note that $\lambda_i(\boldsymbol{x})$ is affine function of $\boldsymbol{x}$ and vanishes on the face F_i. We list the four basic properties of barycentric coordinate below:

1. $0 \le \lambda_i(x) \le 1$;
2. $\sum_{i=1}^{d+1} \lambda_i = 1$;
3. $\lambda_i \in P_1(\tau)$;
4. $\lambda_i(\boldsymbol{x}_j) = \delta_{ij}$.

6.1.2 Shape-regular and quai-uniform triangulations

Given a bounded polyhedral domain $\Omega \subset R^d$. A geometric triangulation (also called mesh or grid) $\mathcal{T}$ of Ω is a set of d-simplices such that

$$\cup\tau = \overline{\Omega}, \quad \text{and} \quad \overset{\circ}{\tau}_i \cap \overset{\circ}{\tau}_j = \emptyset.$$

Examples of triangulations for $\Omega = (0,1)$ $(d=1)$ are shown in Figure 6.1 for $\Omega = (0,1)$ $(d=1)$ and Figure 6.1 for $\Omega = (0,1)^2$ $(d=2)$.

Denote

$$h_\tau = \operatorname{diam}(\tau), h = \max_{\tau\in\mathcal{T}_h} h_\tau; \quad \underline{h} = \min_{\tau\in\mathcal{T}_h} h_\tau,$$

The first requirement is a topological property. A triangulation $\mathcal{T}$ is called *conforming* or *compatible* if the intersection of any two simplexes τ and τ' in $\mathcal{T}$ is either empty or a common lower dimensional simplex (nodes in two dimensions, nodes and edges in three dimensions).

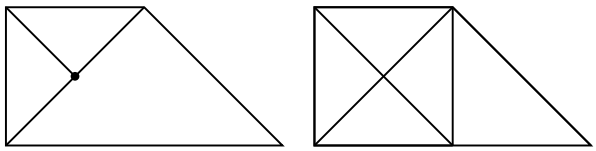


Fig. 6.4. Two triangulations. The left is non-conforming and the right is conforming.

The second important condition depends on the geometric structure. A set of triangulations $\mathscr{T}$ is called *shape regular* if there exists a constant c_0 such that

$$\max_{\tau\in\mathcal{T}} \frac{\operatorname{diam}(\tau)^d}{|\tau|} \le c_0, \quad \forall\, \mathcal{T} \in \mathscr{T}, \tag{6.3}$$

where $\operatorname{diam}(\tau)$ is the diameter of τ and $|\tau|$ is the measure of τ in $\mathbb{R}^d$. This assumption can also be represented as

$$\sup_{h\in\aleph} \max_{\tau\in\mathcal{T}_h} \frac{h_\tau}{\rho_\tau} \le \sigma_1 \tag{6.4}$$

where ρ_τ denotes the radius of the ball inscribed in τ. In two dimensions, it is equivalent to the minimal angle of each triangulation is bounded below uniformly in the shape regular class. We shall define $h_\tau = |\tau|^{1/n}$ for any $\tau \in \mathcal{T} \in \mathscr{T}$. By (6.3), $h_\tau \approx \operatorname{diam}(\tau)$ represents the size of an element $\tau \in \mathcal{T}$ for a shape regular triangulation $\mathcal{T} \in \mathscr{T}$.

In addition to (6.3), if

$$\frac{\max_{\tau\in\mathcal{T}} |\tau|}{\min_{\tau\in\mathcal{T}} |\tau|} \le \rho, \quad \forall\, \mathcal{T} \in \mathscr{T}, \tag{6.5}$$

$\mathscr{T}$ is called *quasi-uniform*. For quasi-uniform grids, $h_{\mathcal{T}} := \max_{\tau\in\mathcal{T}} h_\tau$, the mesh size of $\mathcal{T}$, is used to measure the approximation rate. In the FEM literature, we often write as $\mathcal{T}_h$.

The assumption (6.4) is a local assumption, as is meant by above definition, for $d = 2$ for example, it assures that each triangle will not degenerate into a segment in the limiting case. A triangulation satisfying this assumption is often called to be *shape regular*.

On the other hand, the assumption (6.5) is a global assumption, which says that the smallest mesh size is not too small compared with the largest mesh size of the same triangulation. By the definition, in a quasiuniform triangulation, all the elements are about the same size asymptotically.

Remark 7. In this course, unless otherwise noted, we restrict ourself to quasi-uniform simplicial triangulation. There are other type of meshes by partition the domain into quadrilateral (in 2-D), cubes (in 3-D), or other type of elements.

6.1.3 Finite element space

Given a shape regular triangulation $\mathcal{T}_h$ of Ω, we set

$$V_h := \{v \,|\, v \in C(\overline{\Omega}), \text{ and } v|_\tau \in P_1(\tau), \forall \tau \in \mathcal{T}_h\},$$

where $P_1(\tau)$ denotes the space of polynomials of degree 1 (linear) on $\tau \in \mathcal{T}_h$. Whenever we need to deal with boundary conditions, we further define $V_{h,0} = V_h \cap H_0^1(\Omega)$.

We note here that the global continuity is also necessary in the definition of V_h in the sense that if u has a square interable gradient, that is $u \in H^1(\Omega)$, and u is piecewise smooth, then u is continuous.

We always use n_h to denote the dimension of finite element spaces. For V_h, n_h is the number of vertices of the triangulation $\mathcal{T}_h$ and for $V_{h,0}$, n_h is the number of interior vertices.

Nodal basis functions and dual basis

For linear finite element spaces, we have the so called *a standard nodal basis functions* $\{\varphi_i, i = 1, \cdots n_h\}$ such that φ_i is piecewise linear (with respect to the triangulation) and $\varphi_i(x_j) = \delta_{i,j}$. Note that $\varphi_i|_\tau$ is the corresponding barycentrical coordinates of x_i. See Figure 6.5 for an illustration in 2-D.

Let $(\psi_i)_{i=1}^{n_h}$ be the dual basis of $(\varphi_i)_{i=1}^{n_h}$. Namely

$$(\psi_i, \varphi_j) = \delta_{i,j}, \quad i, j = 1, \ldots, n_h. \tag{6.6}$$

We notice that all the nodal basis functions $\{\varphi_i\}$ are locally supported, but their dual basis functions $\{\psi_i\}$ are in general not locally supported. The nodal basis functions $\{\varphi_i\}$ are easily constructed in terms of barycentric coordinate functions. The dual basis $\{\psi_i\}$ are only interesting for theoretical consideration and it is not necessary to know the actual constructions of these functions.

Therefore for any $v_h \in V_h$, we have the representation

$$v_h(x) = \sum_{i=1}^{n_h} v_h(x_i)\varphi_i(x).$$

Let us see how our construction looks like in one spatial dimension. Associated with the partition $\mathcal{T}_h = \{0 = x_0 < x_1 < \ldots < x_{n_h} < x_{n_h+1} = 1\}$, we define a linear finite element space

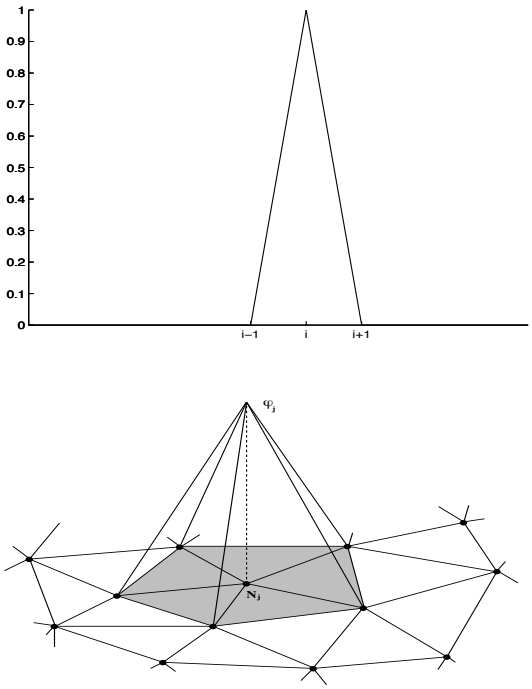


Fig. 6.5. Nodal basis functions in 1d and 2d

$$V_{h,0} = \{v : \ v \text{ is continuous and piecewise linear w. r. t. } \mathcal{T}_h,\ v(0) = v(1) = 0\}.$$

A plot of a typical element of $V_{h,0}$ is shown in Fig. 6.6.

It is easily calculated (as we already mentioned), that the dimension of V_h is equal to the number of internal vertices, and the nodal basis functions spanning $V_{h,0}$ (for $i = 1, 2, \cdots, n_h$) are (see also Fig. 6.5):

$$\varphi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h}, & x \in [x_{i-1}, x_i]; \\ \dfrac{x_{i+1} - x}{h}, & x \in [x_i, x_{i+1}]; \\ 0 & \text{elsewhere.} \end{cases}$$

6.2 Nodal value interpolant

Lemma 19. *For any $v \in V_h$,*

$$v(x) = \sum_{j=1}^{N_h} v(x_j)\phi_j(x).$$

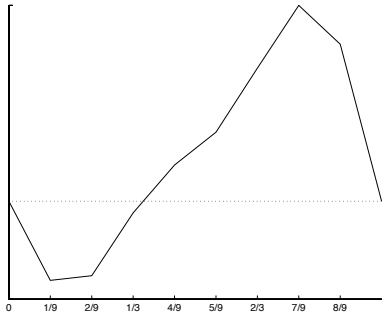


Fig. 6.6. Plot of a typical element from V_h.

Proof. Let $v_h(x) = \sum_{j=1}^{N_h} v(x_j)\phi_j(x)$. For an arbitrary simplex with $d+1$ ($\Omega \subseteq \mathbb{R}^d$) vertices $a_1, \cdots, a_{d+1}$:

$$v_h(x) = \sum_{j=1}^{d+1} v(a_j)\lambda_j(x)$$

Notice that

$$v_h(a_i) = \sum_{j=1}^{d+1} v(a_j)\lambda_j(a_i) = v(a_j),\ i = 1, \cdots, d+1$$

So the values of v_h and v are equal at $d+1$ points. Notice that both v_h and v are linear functions, so $v = v_h$. □

For any continuous function u, we define its finite element interpolation, $u_I \in V_{h,0}$, as follows:

$$u_I(x) = \sum_{i=1}^{N} u(x_i)\phi_i(x). \tag{6.7}$$

For any $v \in \mathcal{S}_0^h$, we can obviously write

$$v(x) = \sum_{i=1}^{N} v(x_i)\phi_i(x).$$

The nodal value interpolation operator $I_h : C(\bar{\Omega}) \mapsto V_h$ is defined as follows

$$(I_h u)(x_i) = u(x_i), \quad \forall\ x_i \in \mathcal{N}_h.$$

6.3 Error estimates

Theorem 10. *Assume that* $\{\mathcal{T}_h : h \in \aleph\}$ *is quasiuniform, then*

$$\inf_{v_h \in V_h} \|v - v_h\| + h|v - v_h|_1 \lesssim h^2|v|_2 \quad \forall\ v \in H^2(\Omega) \tag{6.8}$$

Next we provide proofs of the above theorem for some special cases.

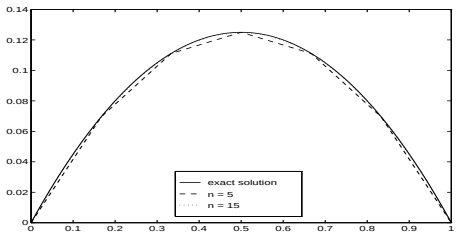


Fig. 6.7. Approximation of finite element space.

A proof of Theorem 10 for $d = 1$

Observe first that $e = (u - u_I)$ vanishes at the end points of each interval and e' is continuous, because e'' is square integrable. By the Rolle's theorem there exists $\xi_i \in (x_i, x_{i+1})$ such that $e'(\xi_i) = 0$. By the Fundamental Theorem of Calculus for $x \in (x_i, x_{i+1})$, we have that

$$e'(x) = \int_{\xi_i}^{x} e''(t)\, dt$$

Since u_I is linear on $[x_i, x_{i+1}]$ we have that $e''(t) = u''(t)$, and hence

$$[e'(x)]^2 = \left[\int_{\xi_i}^{x} u''(t)\, dt\right]^2.$$

Applying the Schwarz inequality to the right side then gives,

$$\begin{aligned}[e'(x)]^2 &\le \left|\int_{\xi_i}^{x} 1^2\, dt\right| \left|\int_{\xi_i}^{x} [u''(t)]^2\, dt\right| \\ &\le |\xi_i - x| \int_{x_i}^{x_{i+1}} [u''(t)]^2\, dt.\end{aligned}$$

Integrating from x_i to x_{i+1}, and observing that

$$\int_{x_i}^{x_{i+1}} |\xi_i - x|\, dx = \frac{1}{2}[(\xi_i - x_i)^2 + (x_{i+1} - \xi_i)^2] \le (x_i - x_{i+1})^2,$$

then gives

$$\int_{x_i}^{x_{i+1}} [e'(x)]^2 \, dx \leq (x_{i+1} - x_i)^2 \int_{x_i}^{x_{i+1}} [u''(t)]^2 \, dt.$$

Finally, summing up on $(0, 1)$ then leads to:

$$\begin{aligned} \|e'\|_{0,\Omega} = \int_0^1 [e'(x)]^2 \, dx &= \sum_{i=1}^{n_h-1} \int_{x_i}^{x_{i+1}} [e'(x)]^2 \, dx \\ &\leq \sum_{i=1}^{n_h-1} (x_{i+1} - x_i)^2 \int_{x_i}^{x_{i+1}} [u''(t)]^2 \, dt \\ &\leq \max_i (x_{i+1} - x_i)^2 \int_0^1 [u''(t)]^2 \, dt. \end{aligned}$$

Since $e(x_i) = 0$, for any $x \in (x_i, x_{i+1})$,

$$|e(x)| = |\int_{x_i}^{x} e'(t) \, dt| \lesssim h|e|_{1,\tau} \lesssim h^2|u|_{2,\tau}.$$

Summing up on $(0, 1)$ then leads to:

$$\|e\|_{0,\Omega} \lesssim h^2|u|_{2,\Omega}.$$

This completes the proof of the estimate in one dimension.

A proof of Theorem 10 for $d = 1, 2, 3$

Let $x = (x^1, \ldots, x^d)$ and $a_i = (a_i^1, \ldots, a_i^d)$. Introducing the auxiliary functions

$$g_i(t) = v(a_i(t)), \text{ with } a_i(t) = a_i + t(x - a_i),$$

we have

$$g_i'(t) = (\nabla v)(a_i(t)) \cdot (x - a_i) = \sum_{l=1}^{d} (\partial_l v)(a_i(t))(x^l - a_i^l)$$

and

$$g_i''(t) = \sum_{k,l=1}^{d} \partial_{kl}^2 v)(a_i(t))(x^k - a_i^k)(x^l - a_i^l). \tag{6.9}$$

By Taylor expansion

$$g_i(0) = g_i(1) - g_i'(1) + \int_0^1 t g_i''(t) dt.$$

Namely

$$v(a_i) = v(x) - (\nabla v)(x) \cdot (x - a_i) + \int_0^1 t g_i''(t) dt. \tag{6.10}$$

Note that

$$(I_h v)(x) = \sum_{i=1}^{n+1} v(a_i)\lambda_i(x), \sum_{i=1}^{n+1} \lambda_i(x) = 1,$$

and

$$\sum_{i=1}^{n+1} (x - a_i)\lambda_i(x) = 0.$$

It follows that

$$(I_h v - v)(x) = \sum_{i=1}^{n+1} \lambda_i(x) \int_0^1 t g_i''(t) dt \tag{6.11}$$

Using (6.9) and the trivial fact that $|x^l - a_i^l| \le h$, we obtain

$$\|g_i''(t)\|_{L^2(\tau)} \le h^2 \sum_{k,l=1}^{d} \|(\partial_{kl}^2 v)(a_i(t))\|_{L^2(\tau_i')} \le h^2 t^{-n/2} \sum_{k,l=1}^{d} \|\partial_{kl}^2 v\|_{L^2(\tau)}$$

where we have used the following change of variable

$$y = a_i + t(x - a_i) : \tau \mapsto \tau_i^t \subset \tau \text{ with } dy = t^d dx.$$

Now taking the $L^2(\tau)$ norm on both hand of sides of (6.11), we get

$$\begin{aligned}
\|I_h v - v\|_{L^2(\tau)} &\le h^2 \sum_{i=1}^{n+1} \max_{x\in\tau} |\lambda_i(x)| \int_0^1 t \|g_i''(t)\|_{L^2(\tau)} \, dt \\
&\le (n+1) \int_0^1 t^{1/n/2} dt \; h^2 \sum_{k,l=1}^{d} \|\partial_{kl}^2 v\|_{L^2(\tau)} \\
&\le \frac{2(n+1)}{4-n} h^2 \sum_{k,l=1}^{d} \|\partial_{kl}^2 v\|_{L^2(\tau)} \\
&\le \frac{4n(n+1)}{4-n} h^2 |v|_{H^2(\tau)}
\end{aligned}$$

Now we prove the H^1 error estimate. Notice that

$$[\partial_j (I_h v - v)](x) = \sum_i (\partial_j \lambda_i)(x) \int_0^1 t g_i''(t) dt + \sum_i \lambda_i(x) \partial_j \int_0^1 t g_i''(t) dt$$

By (6.10),

$$\int_0^1 t g_i''(t) dt = v(a_i) - v(x) + (\nabla v)(x) \cdot (x - a_i)$$

therefore,

$$
\begin{aligned}
&\partial_j \int_0^1 t g_i''(t) dt \\
&= -\partial_j v + (\nabla \partial_j v)(x)(x - a_i) + \nabla v \cdot e_j \quad (e_j \text{ is the } j\text{-th standard basis}) \\
&= (\nabla \partial_j v)(x)(x - a_i)
\end{aligned}
$$

Notice that $\sum_i \lambda_i (\nabla \partial_j v)(x)(x - a_i) = 0$:

$$
[\partial_j (I_h v - v)](x) = \sum_i (\partial_j \lambda_i)(x) \int_0^1 t g_i''(t) dt
$$

Then the estimate for $|\nabla (I_h v - v)|_{L^2(\tau)}$ follows by a similar argument and the following obvious estimate

$$
|(\nabla \lambda_i)(x)| \lesssim \frac{1}{h}.
$$

On the proof of Theorem 12 for $d \geq 4$

The above proof using interpolation for Theorem 12 does not apply for $d \geq 4$. This is because when $d \geq 4$, the embedding relation between $H^2(\Omega)$ and $C(\bar{\Omega})$ is not true. Only continuous functions can have interpolations. In this case, one approach is to use the so-called Scott-Zhang interpolation [17], the details can be found in [24].

Theorem 11. *Let V_N be linear finite element space on a quasi-uniform simplicial triangulation consisting of N element. Then*

$$
\inf_{v_h \in V_N} \|v - v_N\| + h|v - v_N|_1 \lesssim N^{-\frac{2}{d}} |v|_2 \quad \forall\, v \in H^2(\Omega) \tag{6.12}
$$

We can refine and extend the above error estimate in many different ways.

Theorem 12. *Given any function v with certain regularity assumption (say $v \in H^2(\Omega)$), then there is a shape regular grid $\mathcal{T}_N$ consisting of N simplicial elements*

$$
\inf_{v_h \in V_N} \|v - v_N\| + h|v - v_N|_1 \leq C(v) N^{-\frac{2}{d}}. \tag{6.13}
$$

where V_N is linear finite element space on associated with $\mathcal{T}_N$.

Theorem 13. *[2] For any function v that is not locally linear, we have*

$$
\inf_{\dim V_N = N} \inf_{v_h \in V_N} \|v - v_N\| + h|v - v_N|_1 \geq c(v) N^{-\frac{2}{d}}. \tag{6.14}
$$

where V_N be linear finite element space on associated with a shape regular mesh $\mathcal{T}_N$.

7

Deep Neural Network Functions

7.1 Motivation: from finite element to neural network

In this chapter, we will introduce the so-called shallow neural network (deep neural network with one hidden layer) from the viewpoint of finite element method.

Let us first consider the linear finite element functions on the unit interval $\bar{\Omega} = [0, 1]$ in 1D. We then consider a set of equidistant girds Ω_ℓ of level ℓ on the unit interval $\bar{\Omega} = [0, 1]$ and mesh length $h_\ell = 2^{-\ell}$. The grid points $x_{\ell,i}$ are given by

$$x_{\ell,i} := ih_\ell, \quad 0 \le i \le 2^\ell. \tag{7.1}$$

For $\ell = 1$, we denote the special hat function:

$$\varphi(x) = \begin{cases} 2x & x \in [0, \frac{1}{2}] \\ 2(1-x) & x \in [\frac{1}{2}, 1] \\ 0, & \text{others} \end{cases}. \tag{7.2}$$

The next diagram shows this basis function:

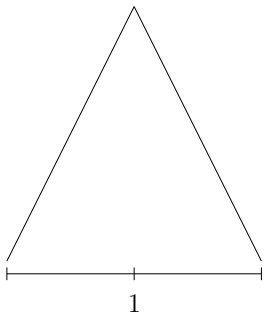


Fig. 7.1. Diagram of $\varphi(x)$

Then, for any nodal basis function $\varphi_{\ell,i}$ as below:

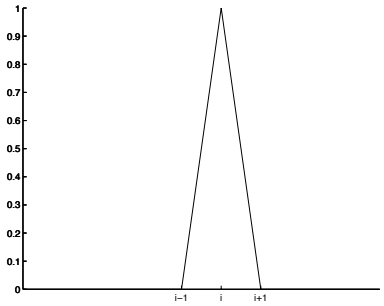


Fig. 7.2. Diagram of $\varphi_{\ell,i}(x)$

in a fine grid $\mathcal{T}_\ell$ can be written as

$$\varphi_{\ell,i} = \varphi(\frac{x - x_{\ell,i-1}}{2h_\ell}) = \varphi(w_\ell x + b_{\ell,i}). \tag{7.3}$$

That is to say, any $\varphi_{\ell,i}(x)$ can be obtained from $\varphi(x)$ ba scaling (dilation) and translation with

$$w_\ell = 2^{\ell-1}, \quad b_{\ell,i} = \frac{-(i-1)}{2}, \tag{7.4}$$

in $\varphi_{\ell,i} = \varphi(w_\ell x + b_{\ell,i})$.

Let recall the finite element interpolation as

$$u(x) \approx u_\ell(x) := \sum_{0 \le i \le 2^\ell} u(x_{\ell,i})\varphi_{\ell,i}(x), \tag{7.5}$$

for any smooth function $u(x)$ on $(0, 1)$. The above interpolation will converge as $\ell \to \infty$, which show that

$$\text{span}\{\varphi(w_\ell x + b_{\ell,i})\} \quad \text{is dense in} \quad H^1(0, 1). \tag{7.6}$$

Thus, we may have the next concise relation:

(7.7)

$$\text{FE space} = \text{span}\left\{\varphi(w_\ell x + b_{\ell,i}) \mid 0 \le i \le 2^\ell, \ell = 1, 2, \cdots\right\} \subset \text{span}\{\varphi(wx + b) \mid w, b \in \mathbb{R}\}.$$

In other words, the finite element space can be understood as the linear combination of $\varphi(wx + b)$ with certain special choice of w and b.

Here, we need to point that this span $\{\varphi(wx + b) \mid w, b \in \mathbb{R}\}$ is exact the deep neural networks with one hidden layer (shallow neural networks) with activation function $\varphi(x)$. More precisely,

$$f \in \text{span}\{\varphi(wx + b) \mid w, b \in \mathbb{R}\}, \tag{7.8}$$

means there exist positive integer N and $w_j, b_j \in \mathbb{R}$ such that

$$f = \sum_{j=1}^{N} a_j \varphi(w_j x + b_j). \tag{7.9}$$

The above function is also called one hidden neural network function with N neurons.

Remark 8. 1. By making w_ℓ and $b_{\ell,i}$ arbitrary, we get a much larger class of function which is exact a special neural network with activation function $\varphi(x)$.

2. Generalizations:
   a) φ can be different, such as $\mathrm{ReLU}(x) = \max\{0, x\}$.
   b) There is a natural extension for hight dimension d as

$$\{\varphi(w \cdot x + b)\}, \tag{7.10}$$

   where $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ and $w \cdot x = \sum_{i=1}^{d} w_i x_i$. This is called "deep" neural network with one hidden layer.

7.2 Why we need deep neural networks via composition

7.2.1 FEM ans DNN_1 in 1D

Thanks to the connection between $\varphi(x)$ and $\mathrm{ReLU}(x) = \{0, x\}$

$$\varphi(x) = 2\mathrm{ReLU}(x) - 4\mathrm{ReLu}(x - \frac{1}{2}) + 2\mathrm{ReLU}(x - 1). \tag{7.11}$$

It suffices to show that each basis function $\varphi_{\ell,i}$ can be represented by a ReLU DNN. We first note that the basis function φ_i has the support in $[x_{i-1}, x_{i+1}]$ can be easily written as

$$\varphi_{\ell,i}(x) = \frac{1}{h_\ell}\mathrm{ReLU}(x - x_{\ell,i-1}) - (\frac{2}{h_\ell})\mathrm{ReLU}(x - x_{\ell,i}) + \frac{1}{h_\ell}\mathrm{ReLU}(x - x_{\ell,i+1}). \tag{7.12}$$

More generally, if function φ_i is not on the uniform grid but has support in $[x_{i-1}, x_{i+1}]$ can be easily written as

$$\varphi_i(x) = \frac{1}{h_{i-1}}\mathrm{ReLU}(x - x_{i-1}) - (\frac{1}{h_{i-1}} + \frac{1}{h_i})\mathrm{ReLU}(x - x_i) + \frac{1}{h_i}\mathrm{ReLU}(x - x_{i+1}), \tag{7.13}$$

where $h_i = x_{i+1} - x_i$.

Thus is to say, we have the next theorem.

Theorem 14. *For $d = 1$, and $\Omega \subset \mathbb{R}^d$ is a bounded interval, then DNN_1 can be used to cover all linear finite element function in on Ω.*

7.2.2 Linear finite element cannot be recovered by DNN_1 for $d \geq 2$

In view of Theorem 14 and the fact that $\mathrm{DNN}_J \subseteq \mathrm{DNN}_{J+1}$, it is natural to ask that how many layers are needed at least to recover all linear finite element functions in $\mathbb{R}^d$ for $d \geq 2$. In this section, we will show that

$$J_d \geq 2, \quad \text{if} \quad d \geq 2, \tag{7.14}$$

where J_d is the minimal J such that all linear finite element functions in $\mathbb{R}^d$ can be recovered by DNN_J.

In particular, we will show the following theorem.

Theorem 15. *If $\Omega \subset \mathbb{R}^d$ is either a bounded domain or $\Omega = \mathbb{R}^d$, DNN_1 can not be used to recover all linear finite element functions on Ω.*

Proof. We prove it by contradiction. Let us assume that for any continuous piecewise linear function $f : \Omega \to \mathbb{R}$, we can find finite $N \in \mathbb{N}$, $w_i \in \mathbb{R}^{1,d}$ as row vector and $\alpha_i, b_i, \beta \in \mathbb{R}$ such that

$$f = \sum_{i=1}^{N} \alpha_i \mathrm{ReLU}(w_i x + b_i) + \beta,$$

with $f_i = \alpha_i \mathrm{ReLU}(w_i x + b_i)$, $\alpha_i \neq 0$ and $w_i \neq 0$. Consider the finite element functions, if this one hidden layer ReLU DNN can recover any basis function of FEM, then it can recover the finite element space. Thus let us assume f is a locally supported basis function for FEM. Furthermore, if Ω is a bounded domain, we assume that

$$d(\mathrm{supp}(f), \partial\Omega) > 0, \tag{7.15}$$

with

$$d(A, B) = \inf_{x \in A, y \in B} \|x - y\|,$$

as the distance of two closed sets.

A more important observation is that $\nabla f : \Omega \to \mathbb{R}^d$ is a piecewise constant vector function. The key point is to consider the discontinuous points for $g := \nabla f = \sum_{i=1}^{N} \nabla f_i$.

For more general case, we can define the set of discontinuous points of a function by

$$D_g := \{x \in \Omega \mid x \text{ is a discontinuous point of } g\}.$$

Because of the property that

$$D_{f+g} \supseteq D_f \cup D_g \backslash (D_f \cap D_g), \tag{7.16}$$

we have

$$D_{\sum_{i=1}^{N} g_i} \supseteq \bigcup_{i=1}^{N} D_{g_i} \backslash \bigcup_{i \neq j} \left(D_{g_i} \cap D_{g_j} \right). \tag{7.17}$$

Note that

$$g_i = \nabla f_i(x) = \nabla\left(\alpha_i \mathrm{ReLU}(w_i x + b_i)\right) = \left(\alpha_i H(w_i x + b_i)\right) w_i \in \mathbb{R}^d, \tag{7.18}$$

for $i = 1 : N$ with H be the Heaviside function defined as:

$$H(x) = \begin{cases} 0 & \text{if } x \le 0, \\ 1 & \text{if } x > 0. \end{cases}$$

This means that

$$D_{g_i} = \{x \mid w_i x + b_i = 0\} \tag{7.19}$$

is a $d-1$ dimensional affine space in $\mathbb{R}^d$.

Without loss of generality, we can assume that

$$D_{g_i} \neq D_{g_j}. \tag{7.20}$$

When the other case occurs, i.e. $D_{g_{\ell_1}} = D_{g_{\ell_2}} = \cdots = D_{g_{\ell_k}}$, by the definition of g_i in (7.18) and D_{g_i} in (7.19) , this happens if and only if there is a row vector (w, b) such that

$$c_{\ell_i}\begin{pmatrix} w & b \end{pmatrix} = \begin{pmatrix} w_{\ell_i} & b_{\ell_i} \end{pmatrix}, \tag{7.21}$$

with some $c_{\ell_i} \neq 0$ for $i = 1 : k$. We combine those g_{ℓ_i} as

$$\begin{aligned} \tilde{g}_\ell &= \sum_{i=1}^k g_{\ell_i} = \sum_{i=1}^k \alpha_{\ell_i} H(w_{\ell_i} x + b_{\ell_i}) w_{\ell_i}, \\ &= \sum_{i=1}^k \left(c_{\ell_i} \alpha_{\ell_i} H\left(c_{\ell_i}(wx + b)\right)\right) w, \\ &= \begin{cases} \left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(c_{\ell_i})\right) w & \text{if} \quad wx + b > 0, \\ \left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(-c_{\ell_i})\right) w & \text{if} \quad wx + b \le 0. \end{cases} \end{aligned}$$

Thus, if

$$\left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(c_{\ell_i})\right) = \left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(-c_{\ell_i})\right),$$

$\tilde{g}_\ell$ is a constant vector function, that is to say $D_{\sum_{i=1}^k g_{\ell_i}} = D_{\tilde{g}_\ell} = \emptyset$. Otherwise, $\tilde{g}_\ell$ is a piecewise constant vector function with the property that

$$D_{\sum_{i=1}^k g_{\ell_i}} = D_{\tilde{g}_\ell} = D_{g_{\ell_i}} = \{x \mid wx + b = 0\}.$$

This means that we can use condition (7.21) as an equivalence relation and split $\{g_i\}_{i=1}^N$ into some groups, and we can combine those g_{ℓ_i} in each group as what we do above. After that, we have

$$\sum_{i=1}^{N} g_i = \sum_{\ell=1}^{\tilde{N}} \tilde{g}_\ell,$$

with $D_{\tilde{g}_s} \neq D_{\tilde{g}_t}$. Finally, we can have that $D_{\tilde{g}_s} \cap D_{\tilde{g}_t}$ is an empty set or a $d-2$ dimensional affine space in $\mathbb{R}^d$. Since $\tilde{N} \leq N$ is a finite number,

$$D := \bigcup_{i=1}^{N} D_{\tilde{g}_\ell} \setminus \bigcup_{s \neq t} \left(D_{\tilde{g}_s} \cap D_{\tilde{g}_t} \right)$$

is an unbounded set.

- If $\Omega = \mathbb{R}^d$,
$$\text{supp(f)} \supseteq D_g = D_{\sum_{i=1}^{N} g_i} = D_{\sum_{\ell=1}^{\tilde{N}} \tilde{g}_\ell} \supseteq D,$$
is contradictory to the assumption that f is locally supported.
- If Ω is a bounded domain,
$$d(D, \partial\Omega) = \begin{cases} s > 0 & \text{if} \quad D_{\tilde{g}_i} \cap \Omega = \emptyset, \forall i \\ 0 & \text{otherwise.} \end{cases}$$
Note again that all $D_{\tilde{g}_i}$'s are $d-1$ dimensional affine spaces, while $D_{\tilde{g}_i} \cap D_{\tilde{g}_j}$ is either an empty set or a d-2 dimensional affine space. If $d(D, \partial\Omega) > 0$, this implies that ∇f is continuous in Ω, which contradicts the assumption that f is a basis function in FEM. If $d(D, \partial\Omega) = 0$, this contradicts the previous assumption in (7.15).

Hence DNN_1 cannot recover any piecewise linear function in Ω for $d \geq 2$. □

Following the proof above, we have the following theorem.

Theorem 16. *$\{\text{ReLU}(w_i x + b_i)\}_{i=1}^{m}$ are linearly independent if (w_i, b_i) and (w_j, b_j) are linearly independent in $\mathbb{R}^{1\times(d+1)}$ for any $i \neq j$.*

7.3 Definition of neural network space

1. Primary variables $n_0 = d$

$$x^0 = x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

2. n_1 hyperplanes

$$W^1 x + b^1 = \begin{pmatrix} w_1^1 x + b_1^1 \\ w_2^1 x + b_2^1 \\ \vdots \\ w_n^1 x + b_n^1 \end{pmatrix}$$

3. n_1-neurons:

$$x^1 = \sigma(W^1 x + b^1) = \begin{pmatrix} \sigma(w_1^1 x + b_1^1) \\ \sigma(w_2^1 x + b_2^1) \\ \vdots \\ \sigma(w_n^1 x + b_n^1) \end{pmatrix}$$

4. n_2-hyperplanes

$$W^2 x^1 + b^2 = \begin{pmatrix} w_1^2 x^1 + b_1^2 \\ w_2^2 x^1 + b_2^2 \\ \vdots \\ w_n^2 x^1 + b_n^2 \end{pmatrix}$$

Shallow neural network functions:

(7.22)

$${}_n\mathrm{N}(n_1, n_2) = {}_n\mathrm{N}(\sigma; n_1, n_2) = \left\{ W^2 x^1 + b^2, x^1 = \sigma(W^1 x + b^1) \text{ with } W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, b^\ell \in \mathbb{R}^{n_\ell}, \ell = 1, 2, n_0 = d \right\}$$

(7.23)

$${}_n\mathrm{N}(\sigma; n_1, n_2, \ldots, n_L) = \left\{ W^L x^{L-1} + b^L, x^\ell = \sigma(W^\ell x^{\ell-1} + b^\ell) \text{ with } W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, b^\ell \in \mathbb{R}^{n_\ell}, \ell = 1 : L, n_0 = d, x^0 = x \right\}$$

First, let us first define the so-called 1-hidden layer (shallow) neural network.

The 1-hidden layer (shallow) neural network is defined as:

(7.24)
$$ {}_n\mathrm{N} = {}_n\mathrm{N}(\sigma) = {}_n\mathrm{N}^1(\sigma) = \bigcup_{n_1 \geq 1} {}_n\mathrm{N}(\sigma; n_1, 1)$$

The 2-hidden layer (shallow) neural network is defined as:

(7.25)
$$ {}_n\mathrm{N}^2(\sigma) = \bigcup_{n_1, n_2 \geq 1} {}_n\mathrm{N}(\sigma; n_1, n_2, 1)$$

7.4 Qualitative convergence results

The first question about DNN_1 is about the approximation properties for any continuous functions. Here we have the next theorem.

The first proof for this lemma above can be found in [14] and summarized in [16]. The next theorem plays an important role in the proof of above lemma, which is first proved in [14] with several steps. Here we can present a more direct and simple version.

Theorem 17 (Universal Approximation Property of Shallow Neural Networks). *Let σ be a Riemann integrable function and $\sigma \in L^\infty_{loc}(\mathbb{R})$. Then $\Sigma_d(\sigma)$ in dense in $C(\Omega)$ for any compact $\Omega \subset \mathbb{R}^n$ if and and only if σ is not a polynomial!*

Namely, if σ is not a polynomial, then, for any $f \in C(\bar{\Omega})$, there exists a sequence $\phi_n \in \mathrm{DNN}_1$ such that

$$\max_{x\in\bar{\Omega}} |\phi_n(x) - f(x)| \to 0, \quad n \to \infty.$$

Proof. Let us first prove the theorem in a special case that $\sigma \in C^\infty(\mathbb{R})$. Since $\sigma \in C^\infty(\mathbb{R})$, it follows that for every ω, b,

$$\frac{\partial}{\partial \omega_j}\sigma(\omega \cdot x + b) = \lim_{n\to\infty} \frac{\sigma((\omega + he_j)\cdot x + b) - \sigma(\omega \cdot x + b)}{h} \in \overline{\Sigma}_d(\sigma) \tag{7.26}$$

for all $j = 1, ..., d$.

By the same argument, for $\alpha = (\alpha_1, ..., \alpha_d)$

$$D^\alpha_\omega \sigma(\omega \cdot x + b) \in \overline{\Sigma}_d(\sigma)$$

for all $k \in \mathbb{N}$, $j = 1, ..., d$, $\omega \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

Now

$$D^\alpha_\omega \sigma(\omega \cdot x + b) = x^\alpha \sigma^{(k)}(\omega \cdot x + b)$$

where $k = |\alpha|$ and $x^\alpha = x_1^{\alpha_1} \cdots x_d^{\alpha_d}$. Since σ is not a polynomial there exists a $\theta_k \in \mathbb{R}$ such that $\sigma^{(k)}(\theta_k) \neq 0$. Taking $\omega = 0$ and $b = \theta_k$, we thus see that $x_j^k \in \overline{\Sigma}_d(\sigma)$. Thus, all polynomials of the form $x_1^{k_1} \cdots x_d^{k_d}$ are in $\overline{\Sigma}_d(\sigma)$.

This implies that $\overline{\Sigma}_d(\sigma)$ contains all polynomials. By Weierstrass's Theorem [19] it follows that $\overline{\Sigma}_d(\sigma)$ contains $C(K)$ for each compact $K \subset \mathbb{R}^n$. That is $\Sigma_d(\sigma)$ is dense in $C(\mathbb{R}^d)$.

Now we consider the case that σ is only Riemann integrable. Consider the mollifier η

$$\eta(x) = \frac{1}{\sqrt{\pi}} e^{-x^2}.$$

Set $\eta_\epsilon = \frac{1}{\epsilon}\eta(\frac{x}{\epsilon})$. Then consider σ_{η_ϵ}

$$\sigma_{\eta_\epsilon}(x) := \sigma * \eta_\epsilon(x) = \int_{\mathbb{R}} \sigma(x - y)\eta_\epsilon(y)dy \tag{7.27}$$

It can be seen that $\sigma_{\eta_\epsilon} \in C^\infty(\mathbb{R})$. We first notice that $\overline{\Sigma}_1(\sigma_{\eta_\epsilon}) \subset \overline{\Sigma}_1(\sigma)$, which can be done easily by checking the Riemann sum of $\sigma_{\eta_\epsilon}(x) = \int_{\mathbb{R}} \sigma(x-y)\eta_\epsilon(y)dy$ is in $\overline{\Sigma}_1(\sigma)$.

Following the argument in the beginning of the proof proposition, we want to show that $\overline{\Sigma}_1(\sigma_{\eta_\epsilon}))$ contains all polynomials. For this purpose, it suffices to show that there exists θ_k and σ_{η_ϵ} such that $\sigma^{(k)}_{\eta_\epsilon}(\theta_k) \neq 0$ for each k. If not, then there must be k_0 such that $\sigma^{(k_0)}_{\eta_\epsilon}(\theta) = 0$ for all $\theta \in \mathbb{R}$ and all $\epsilon > 0$. Thus σ_{η_ϵ}'s are all polynomials with degree at most $k_0 - 1$. In particular, It is known that $\eta_\epsilon \in C^\infty_0(\mathbb{R})$ and $\sigma * \eta_\epsilon$ uniformly converges to σ on compact sets in $\mathbb{R}$ and $\sigma * \eta_\epsilon$'s are all polynomials of degree at most $k_0 - 1$. Polynomials of a fixed degree form a closed linear subspace, therefore σ is also a polynomial of degree at most $k_0 - 1$, which leads to contradiction. □

Properties of polynomials using Fourier transform

We make use of the theory of tempered distributions (see [20] for an introduction) and we begin by collecting some results of independent interest, which will also be important later. We begin by noting that an activation function σ which satisfies a polynomial growth condition $|\sigma(x)| \leq C(1 + |x|)^n$ for some constants C and n is a tempered distribution. As a result, we make this assumption on our activation functions in the following theorems. We briefly note that this condition is sufficient, but not necessary (for instance an integrable function need not satisfy a pointwise polynomial growth bound) for σ to be represent a tempered distribution.

We begin by studying the convolution of σ with a Gaussian mollifier. Let η be a Gaussian mollifier

$$\eta(x) = \frac{1}{\sqrt{\pi}} e^{-x^2}. \tag{7.28}$$

Set $\eta_\epsilon = \frac{1}{\epsilon}\eta(\frac{x}{\epsilon})$. Then consider σ_ϵ

$$\sigma_\epsilon(x) := \sigma * \eta_\epsilon(x) = \int_{\mathbb{R}} \sigma(x-y)\eta_\epsilon(y)dy \tag{7.29}$$

for a given activation function σ.

It is clear that $\sigma_\epsilon \in C^\infty(\mathbb{R})$. Moreover, by considering the Fourier transform (as a tempered distribution) we see that

$$\hat{\sigma}_\epsilon = \hat{\sigma}\hat{\eta}_\epsilon = \hat{\sigma}\eta_{\epsilon^{-1}}. \tag{7.30}$$

We begin by stating a lemma which characterizes the set of polynomials in terms of their Fourier transform.

Lemma 20. *Given a tempered distribution σ, the following statements are equivalent:*

1. *σ is a polynomial*
2. *σ_ϵ given by (7.29) is a polynomial for any $\epsilon > 0$.*
3. *$\mathrm{supp}(\hat{\sigma}) \subset \{0\}$.*

Proof. We begin by proving that (3) and (1) are equivalent. This follows from a characterization of distributions supported at a single point (see [20], section 6.3). In particular, a distribution supported at 0 must be a finite linear combination of Dirac masses and their derivatives. In particular, if $\hat{\sigma}$ is supported at 0, then

$$\hat{\sigma} = \sum_{i=1}^{n} a_i \delta^{(i)}. \tag{7.31}$$

Taking the inverse Fourier transform and noting that the inverse Fourier transform of $\delta^{(i)}$ is $c_i x^i$, we see that σ is a polynomial. This shows that (3) implies (1), for the converse we simply take the Fourier transform of a polynomial and note that it is a finite linear combination of Dirac masses and their derivatives.

Finally, we prove the equivalence of (2) and (3). For this it suffices to show that $\hat{\sigma}$ is supported at 0 iff $\hat{\sigma}_\epsilon$ is supported at 0. This follows from equation 7.30 and the fact that $\eta_{\epsilon^{-1}}$ is nowhere vanishing. □

As an application of Lemma 20, let us give a simple proof of the following result. The first proof of this result can be found in [14] and is summarized in [16]. Extending this result to the case of non-smooth activation is first done in several steps in [14]. Our contribution is to provide a much simpler argument based on Fourier analysis.

8

Convolutional Multigrid Method

8.1 A one dimensional problem

8.1.1 Two-point boundary problems and finite element discretization

Define the functional space

$$V = \{v : [0,1] \to R,\ v \text{ is continuous and } v(0) = v(1) = 0\}.$$

Given any $f : [0,1] \to R$, consider

$$J(v) = \frac{1}{2}\int_0^1 |v'|^2 dx - \int_0^1 fv dx.$$

Find $u \in V$ such that

$$u = \arg\min_{v \in V} J(v) \tag{8.1}$$

Theorem 18. *Problem* (8.1) *is equivalent to: Find $u \in V$ such that*

$$\begin{cases} -u'' = f,\ 0 < x < 1, \\ u(0) = u(1) = 0. \end{cases} \tag{8.2}$$

Proof. For any $v \in V, t \in R$, let $g(t) = J(u + tv)$. Since $u = \arg\min_{v \in V} J(v)$ means $g(t) \geq g(0)$. Hence, for any $v \in V$, 0 is the global minimum of the function $g(t)$. Therefore $g'(0) = 0$ implies

$$\int_0^1 u'v' dx = \int_0^1 fv dx \quad \forall v \in V.$$

By integration by parts, which is equivalent to

$$\int_0^1 (-u'' - f)v dx = 0 \quad \forall v \in V.$$

It can be proved that the above identity holds if and only if $-u'' = f$ for all $x \in (0,1)$. Namely u satisfies (8.10). □

Let V_h be finite element space and $\{\varphi_1, \varphi_2, \cdots \varphi_n\}$ be a nodal basis of the V_h (see Section 6.1). Let $\{\psi_1, \psi_2, \cdots, \psi_n\}$ be a dual basis of $\{\varphi_1, \varphi_2, \cdots \varphi_n\}$, namely $(\varphi_i, \psi_j) = \delta_{ij}$.

$$J(v_h) = \frac{1}{2}\int_0^1 |v_h'|^2 dx - \int_0^1 f v_h dx. \tag{8.3}$$

Let

$$v_h = \sum_{i=1}^n v_i \varphi_i,$$

then

$$J(v_h) = I(v) = \frac{1}{2} v^T A * v - b^T v$$

and

$$\nabla I(v) = A * v - b.$$

At the same time, let $u_h = \sum_{i=1}^n \mu_i \varphi_i$,

$$u_h = \arg\min_{v_h \in V_h} J(v_h) \Leftrightarrow \mu = \arg\min_{v \in R^n} I(v) \tag{8.4}$$

And u_h solves the problem: Find $u_h \in V_h$

$$\frac{d}{dt} J(u_h + t v_h)|_{t=0} = a(u_h, v_h) - \langle f, v_h \rangle = 0 \quad \forall v_h \in V_h. \tag{8.5}$$

where

$$a(u_h, v_h) = \int_0^1 u_h' v_h' dx.$$

which is equivalent to solving $\underset{\sim}{A}\mu = b$, where $\underset{\sim}{A} = (a_{ij})_{ij}^n$ and $a_{ij} = a(\varphi_j, \varphi_i)$ and $b_i = \int_0^1 f\varphi_i dx$. Namely

$$\frac{1}{h}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \\ \mu_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{pmatrix}. \tag{8.6}$$

Which can be rewritten as

$$\frac{-\mu_{i-1} + 2\mu_i - \mu_{i+1}}{h} = b_i, \quad 1 \le i \le n, \quad \mu_0 = \mu_{n+1} = 0. \tag{8.7}$$

Using the convolution notation, (8.7) can be written as

$$A * \mu = b, \tag{8.8}$$

where $A = \frac{1}{h}[-1, 2, -1]$.

8.1.2 Spectrum properties of $A*$

We recall that λ is an eigenvalue of A and and $\xi \in R^N \setminus \{0\}$ is a corresponding eigenvector if

$$A * \xi = \lambda \xi.$$

Because of the special structure of A, all the N eigenvalues, λ_k, and the corresponding eigenvectors, $\xi^k = (\xi^k_j)$, of A can be obtained, for $1 \le k \le N$, as follows:

$$\lambda_k = \frac{4}{h} \sin^2 \frac{k\pi}{2(N+1)}, \quad \xi^k_j = \sin \frac{kj\pi}{N+1} (1 \le j \le N).$$

Indeed, the relation $A\xi^k = \lambda_k \xi^k$ can be verified by following elementary trigonometric identities:

$$-\sin \frac{k(j-1)\pi}{N+1} + 2 \sin \frac{kj\pi}{N+1} - \sin \frac{k(j+1)\pi}{N+1} = 4 \sin^2 \frac{k\pi}{2(N+1)} \sin \frac{kj\pi}{N+1},$$

where $1 \le j \le N$. Actually, it is not very difficult to derive these formula directly (see appendix A).

To understand the behavior of these eigenvectors, let us take $N = 6$ and plot the linear interpolations of all these 6 eigenvectors. We immediately observe that each vector ξ^k corresponds to a given frequency, and larger k corresponds to higher frequency. As λ_k is increasing with respect to k, we can then say that a larger eigenvalue of A corresponds to a higher frequency eigenvector. From a numerical point of view, we say a relatively low frequency vector is relatively smoother whereas a high frequency vector is nonsmooth.

We note that the set of eigenvectors $\xi^k = (\xi^k_j)$ forms an orthogonal basis of R^N. (This fact can be checked directly, or it also follows from the fact that the matrix A is symmetric and has N distinctive eigenvalues.) Therefore, any $\xi \in R^N$ can be expanded in terms of these eigenvectors:

$$\xi = \sum_{k=1}^{N} \alpha_k \xi^k.$$

This type of expansion is often called discrete Fourier expansion. The smoothness of the vector ξ has a lot to do with the relative size of the coefficients α_k. To see this numerically, let us again take $N = 4$ and consider the following two vectors

$$\xi = \sum_{k=1}^{4} 2^{1-k} \xi^k, \quad \sigma = \sum_{k=1}^{4} 2^{k-4} \xi^k.$$

The first vector ξ has larger coefficients in front of lower frequencies whereas the second vector σ has larger coefficients in front of higher frequencies. From Figure 8.2, it is easy to see how the smoothness of a vector depends on the relative size of its Fourier coefficients. We conclude that, in general, a vector with relatively small Fourier coefficients in front of the higher frequencies is relatively smooth and conversely, a vector with relatively large Fourier coefficients in front of the higher frequencies is relatively rough or nonsmooth.

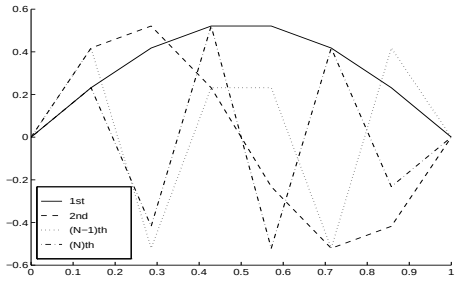


Fig. 8.1. The eigenvectors

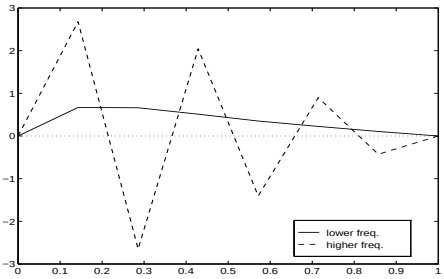


Fig. 8.2. Plots of ξ and σ. ξ-solid line; σ dashed line

8.1.3 Gradient descent method

Noting that $\nabla I(v) = A * v - b$ and applying the gradient descent method to solve problem (8.4), we obtain

$$\mu^{(l)} = \mu^{(l-1)} - \eta(A * \mu^{(l-1)} - b), \quad l = 1, \cdots, \nu.$$

After ν iterations of gradient descent method, we denote the solution as u_h^ν.

Consider the finite element discretization of Poisson equation in 1D: One very simple iterative method for (8.6) is the following gradient descent method

$$\mu^{(l)} = \mu^{(l-1)} + \eta(b - A * \mu^{(l-1)}),$$

or, for $j = 1 : N$,

$$\mu_j^{(l)} = \mu_j^{(l-1)} + \eta\left(\beta_j - \frac{-\mu_{j-1}^{(l-1)} + 2\mu_j^{(l-1)} - \mu_{j+1}^{(l-1)}}{h}\right),$$

where $\eta > 0$ is a positive parameter named learning rate.

It is not so difficult to properly choose η so that the above iterative scheme converges, namely for any initial guess μ^0, the sequence $(\mu^(l-1))$ generated by the above iteration converges to the exact solution μ of (8.6):

Note that

$$\mu = \mu + \eta(b - A * \mu),$$

we get

$$\mu - \mu^{(l)} = (I - \eta A*)(\mu - \mu^{(l-1)}),$$

or

$$\mu - \mu^{(l)} = (I - \eta A*)^l(\mu - \mu^0), l = 1, 2, 3, \cdots$$

As we known,

$$(I - \eta A*)^l \longrightarrow o$$

if and only if $\rho(I - \eta A*) < 1$. Here $\rho(B)$ is the spectral radius of matrix B. However, $\rho(I - \eta A*) < 1$ if and only if

$$0 < \text{all the eigenvalue of } A* < 2\eta^{-1}.$$

Thus, a necessary and sufficient condition for the convergence is the following

$$0 < \eta < \frac{2}{\rho(A*)}.$$

It is easy to see that (for example, $4/h$ is an upper bound of its row sums)

$$\frac{4}{h} > \rho(A*)$$

Therefore it is reasonable to make the following choice:

$$\eta = \frac{h}{4}$$

and the resulting algorithm is

$$\mu^{(l)} = \mu^{(l-1)} + \frac{h}{4}(b - A * \mu^{(l-1)}). \tag{8.9}$$

In the rest of this section, unless otherwise noted, we shall choose η as above for simplicity.

On Figure 8.3 the convergence history plot of the above gradient descent iterative method for typical application is shown. As we see, this iterative scheme converges very slowly.

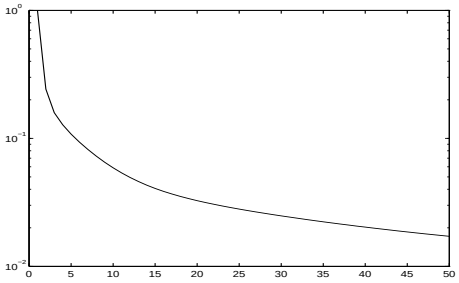


Fig. 8.3. A picture on the GD method convergence history

Our main goal is to find a way to speed up such kind of rather slowly convergent iterative scheme. To do that, we need to study its convergent property in more microscopic level. First of all, let us now take a careful look at the convergence history picture and make the following observation:

> *Observation 1.* The scheme converges rather fast in the very beginning but then slows down after a few steps. Overall, the method converges very slowly.

To further understand this phenomenon, let us plot the detailed pictures of the error functions in the first few iterations. After a careful look at these pictures, we have the following observation:

Observation 2. The scheme not only converges fast in the first few steps, but also smooth out the error function very quickly.

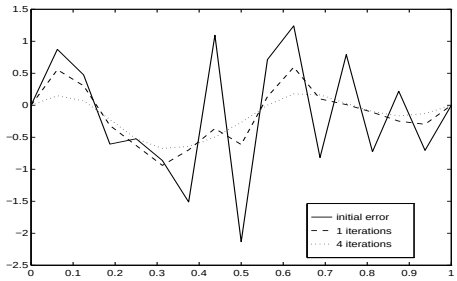


Fig. 8.4. The smoothing effect of the Richardson method

In other words, the error function becomes a much smoother function after a few such simple iterations. This property of the iterative scheme is naturally called *a smoothing property* and an iterative scheme having this smoothing property is called a *smoother*.

The above two observations, especially the second one, concern the most important property of the simple gradient descent method that we can take advantage to get a much faster algorithm.

Example 6. Let $f(x) = \pi^2 \sin \pi x$. Consider

$$\begin{cases} -u'' = f, \ 0 < x < 1, \\ u(0) = u(1) = 0. \end{cases} \tag{8.10}$$

The true solution $u = \sin \pi x$. Given the partition with the grid points $x_i = \frac{i}{n+1}, i = 0, 1, \cdots, n+1$, then by finite element discretization, we obtain

$$A * \mu = b, A = \frac{1}{h}[-1, 2, -1]. \tag{8.11}$$

Use gradient descent method to solve (8.11) with random initial guess μ^0. Plot $\mu^\ell - \mu^0$ and $\|\mu^\ell - \mu^0\|$ for $\ell = 1, 2, 3$.

The gradient descent method can be written in terms of $S_0^\ell : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_\ell}$ satisfying

$$\mu^{(1)} = (S_0^\ell b) = \frac{h_\ell}{4} b, \tag{8.12}$$

for equation (8.7) with initial guess zero. If we apply this method twice, then

$$\mu^{(2)} = S_1^\ell(b) = S_0^\ell b + S_0^\ell(b - A_\ell * (S_0^\ell b)),$$

with element-wise form

$$\mu_i^{(2)} = \frac{h_\ell}{16}(b_{i-1} + 6b_i + b_{i+1}) \tag{8.13}$$

Then by the definition of convolution (9.17), we have

$$\mu^{(1)} = S_0^\ell * b \quad \mu^{(2)} = S_1^\ell * b. \tag{8.14}$$

with

$$S_0^\ell = \frac{h_\ell}{4}, \tag{8.15}$$

and

$$S_1^\ell = \frac{h_\ell}{16}[1, 6, 1]. \tag{8.16}$$

Hence we denote S_0^ℓ or S_1^ℓ as S^ℓ.
Now for any given $\mu^{(0)} = \tilde{\mu}^{(0)}$,

$$\begin{aligned} &j = 1, 2, \cdots, 2\nu \\ &\mu^{(j)} = \mu^{(j-1)} + S_0^\ell * (b - A * \mu^{(j-1)}) \end{aligned} \tag{8.17}$$

$$\Leftrightarrow$$

$$\begin{aligned} &j = 1, 2, \cdots, \nu \\ &\tilde{\mu}^{(j)} = \tilde{\mu}^{(j-1)} + S_1^\ell * (b - A * \tilde{\mu}^{(j-1)}) \end{aligned} \tag{8.18}$$

we obtain $\mu^{(j)} = \tilde{\mu}^{(j)}$ which means one step S_1^ℓ is equivalent to two steps of S_0^ℓ .

8.1.4 Convergence and smoothing properties of GD

Because of the extraordinary importance of this smoothing property, we shall now try to give some simple theoretical analysis. To do this, we make use of the eigenvalues and eigenvectors of the matrix A.

Fourier analysis for the gradient descent method

Our earlier numerical experiments indicate that the gradient descent method has a smoothing property. Based on our understanding of the relation between the smoothness and the size of Fourier coefficients, we can imagine that this smoothing property can be analyzed using the discrete Fourier expansion.

Let μ be the exact solution of (8.6) and $\mu^{(l)}$ the result of $l-th$ iteration from the gradient descent method (8.9). Then

$$\mu - \mu^{(l)} = (1 - \eta A*)(\mu - \mu^{(l-1)}) = \ldots = (1 - \eta A*)^l(\mu - \mu^{(0)}).$$

Consider the Fourier expansion of the initial error:

$$\mu - \mu^{(0)} = \sum_{k=1}^{N} \alpha_k \xi^k.$$

Then

$$\mu - \mu^{(l)} = \sum_{k=1}^{N} \alpha_k (I - \eta A*)^l \xi^k.$$

Note that $\eta = h/4$ and for any polynomial p

$$p(A*)\xi^k = p(\lambda_k)\xi^k,$$

we get

$$\mu - \mu^{(m)} = \sum_{k=1}^{N} \alpha_k (1 - \eta\lambda_k)^m \xi^k = \sum_{k=1}^{N} \alpha_k^{(m)} \xi^k$$

where

$$\alpha_k^{(m)} = \left(1 - \sin^2 \frac{k\pi}{2(N+1)}\right)^m \alpha_k.$$

Note that

$$1 - \sin^2 \frac{k\pi}{2(N+1)} = \cos^2 \frac{k\pi}{2(N+1)} = \sin^2(\frac{\pi}{2} - \frac{k\pi}{2(N+1)})$$

implies

$$\alpha_k^{(m)} = \alpha_k \sin^{2m} \frac{N+1-k}{N+1}\frac{\pi}{2} \leq \alpha_k \left(\frac{N+1-k}{N+1}\frac{\pi}{2}\right)^{2m}$$

which, for k close to N, for example $k = N$, approaches to 0 very rapidly when $m \to \infty$. This means that high frequency components get damped very quickly. However, for k far away from N, for example $k = 1$, $\alpha_k^{(m)}$ approaches to 0 very slowly when $m \to \infty$.

This simple analysis clearly justifies the smoothing property that has been observed by numerical experiments.

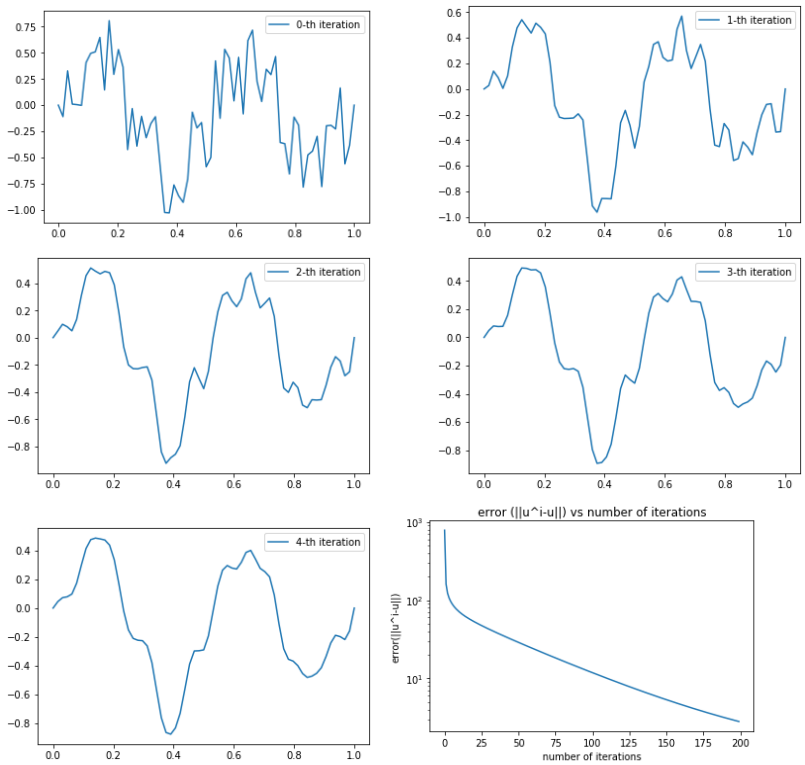


Fig. 8.5. u^0, u^1, u^2, u^3, u^4

An intuitive discussion

Both the gradient decsent and Gauss-Seidel methods are oftentimes called *local relaxation* methods. This name refers to the fact that what both of these algorithms do are just trying to correct the residual vector locally at one nodal point at a time (recall that $\mu_j \approx u(x_j)$). This local relaxation procedure is then effective to the error components that are local in nature. Incidently, the nonsmooth or high frequency component which oscillates across one or few grid points have a strong local feature. Therefore, it is not surprising the both gradient descent and Gauss-Seidel iteration can damp out these nonsmooth components more easily. These methods are very inefficient for relatively smoother components in the error since a smoother function is more globally related in nature.

8.1.5 Coarse grid correction and two grid method

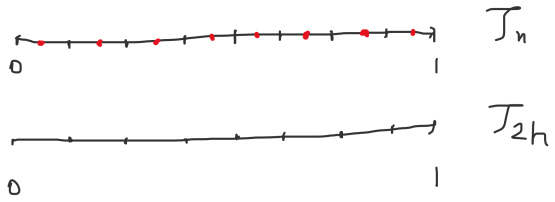


Fig. 8.6. Two level grids.

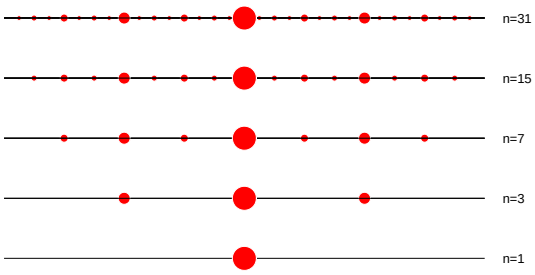


Fig. 8.7. Multiple grids in one dimension

As we discussed earlier, although gradient descent iteration usually converges very slowly, it does quickly smooth out the rough component in the error. In other words, the error becomes smooth after a few gradient descent iterations on a fine grid, but looks rough when viewed on a coarser grid. Hence, a few gradient descent iterations can further reduce the error on the coarse grid. The main idea of two grid method or multigrid method is to use the fact that a smooth function can be well approximated on a coarser grid.

In summery, we can write the two grid method in terms of FE functions as follows:

Algorithm 7 A two grid method (in terms of FE functions)

Input u^0.

Step 1: Apply ν_1-times gradient descent iterations for

$$\min_{v^1 \in V_1} J(v^1)$$

with initial guess u^0 to obtain $u^1 \in V_1$.

Step 2: Apply ν_2-times gradient descent iterations for

$$\min_{v^2 \in V_2} J(u^1 + v^2)$$

with zero initial guess to obtain $u^2 \in V_2$.

Step 3: Update: $u = u^1 + u^2$.

Realization of step 1:

Step 1: Given $u^0 \in V_1$, apply ν_1-times gradient descent method for

$$\min_{v^1 \in V_1} J(v^1)$$

with initial guess u^0 to obtain $u^1 \in V_1$.

Let $u^0 = \sum_{i=1}^{n_1} \mu_i^0 \phi_i^1$, $\quad v^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1$, $\quad \mu^0 = \{\mu_i^0\}_{i=1}^{n_1}$, $\quad \mu^1 = \{\mu_i^1\}_{i=1}^{n_1}$,
where $\phi^1 = (\phi_1^1, \phi_2^1, \cdots, \phi_{n_1}^1)$ is the nodal basis of V_1.
Namely, $b^1 = b, \mu^1 \leftarrow \mu^0$, for $i = 1 : \nu_2$

$$\mu^1 \leftarrow \mu^1 - \eta_1(A_1 * \mu^1 - b^1).$$

After ν_1 iterations, we obtain updated μ^1 and $u^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1$.

Realization of step 2:

Step 2: Apply ν_2-times gradient descent iterations for

$$\min_{v^2 \in V_2} J(u^1 + v^2)$$

with zero initial guess to obtain $u^2 \in V_2$.

Let

$$u^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1,\ v^2 = \sum_{i}^{n_2} \mu_i^2 \phi_i^2,\ \mu^1 = \{\mu_i^1\}_{i=1}^{n_1},\ \mu^2 = \{\mu_i^2\}_{i=1}^{n_2}.$$

We have

$$\begin{aligned} J(u^1+v^2) &= \frac{1}{2}\int_0^1 |(u^1+v^2)'|^2 dx - \int_0^1 f(u^1+v^2)dx \\ &= J(u^1)+J(v^2)+\int_0^1 (u^1)'(v^2)'dx \\ &= \frac{1}{2}(\mu^1)^T A_1 * \mu^1 + \frac{1}{2}(\mu^2)^T A_2 * \mu^2 - (\mu^2)^T r^2 \end{aligned} \tag{8.19}$$

where

$$r_i^2 = \int_0^1 f\phi_i^2 - (u^1)'(\phi_i^2)'dx = (f,\phi_i^2) - a(u^1,\phi_i^2). \tag{8.20}$$

Now noting that

$$\phi_i^2 = \frac{1}{2}\phi_{2i-1}^1 + \phi_{2i}^1 + \frac{1}{2}\phi_{2i+1}^1, \tag{8.21}$$

Let $\phi^2 = \{\phi_i^2\}_{i=1}^{n_2}, \phi^1 = \{\phi_i^1\}_{i=1}^{n_1}$. Using the convolution with stride notation, we obtain

$$\phi^2 = R *_2 \phi^1 \tag{8.22}$$

with $R = [\frac{1}{2}, 1, \frac{1}{2}]$. Futhermore,

$$\begin{aligned} r_i^2 &= (f, \frac{1}{2}\phi_{2i-1}^1 + \phi_{2i}^1 + \frac{1}{2}\phi_{2i+1}^1) - a(u^1, \frac{1}{2}\phi_{2i-1}^1 + \phi_{2i}^1 + \frac{1}{2}\phi_{2i+1}^1) \\ &= \frac{1}{2}b_{2i-1}^1 + b_{2i}^1 + \frac{1}{2}b_{2i+1}^1 - \Big(\frac{1}{2}(A_1*\mu^1)_{2i-1} + (A_1*\mu^1)_{2i} + \frac{1}{2}(A_1*\mu^1)_{2i+1}\Big) \\ &= \frac{1}{2}(b^1 - A_1*\mu^1)_{2i-1} + (b^1 - A_1*\mu^1)_{2i} + \frac{1}{2}(b^1 - A_1*\mu^1)_{2i+1} \\ &= \frac{1}{2}r_{2i-1}^1 + r_{2i}^1 + \frac{1}{2}r_{2i+1}^1, \end{aligned} \tag{8.23}$$

where $r^1 = b^1 - A_1 * \mu^1$.

Lemma 21. *Using the convolution with stride notation, we have*

$$r^2 = R *_2 r^1$$

with $R = [\frac{1}{2}, 1, \frac{1}{2}]$.

Therefore applying gradient descent method ν_2-times for (8.34) reads:

$r^1 = b^1 - A_1 * \mu^1, r^2 = R *_2 r^1, \mu^2 \leftarrow 0$, for $i = 1 : \nu_2$

$$\mu^2 \leftarrow \mu^2 - \eta_2(A_2 * \mu^2 - r^2). \tag{8.24}$$

After ν_2 iterations, we obtain updated μ^2 and $u^2 = \sum_{i=1}^{n_2} \mu_i^2 \phi_i^2$.

Realization of step 3:

Step 3: $u = u^1 + u^2$.

Let $\mu^2 = \{\mu_i^2\}_{i=1}^{n_2}, \phi^2 = \{\phi_i^2\}_{i=1}^{n_2}$. Therefore $u^2 = \sum_{i=1}^{n_2} \mu_i^2 \phi_i^2 = (\mu^2, \phi^2)_{l^2}$ and by (8.22) we have

$$\begin{aligned} u^2 &= (\mu^2, \phi^2)_{l^2} = (\mu^2, R *_2 \phi^1)_{l^2} = (R *_2^\top \mu^2, \phi^1)_{l^2} \\ &= \sum_{i=1}^{n_1} \left(R *_2^\top \mu^2\right)_i \phi_i^1 \end{aligned} \tag{8.25}$$

with $R = [\frac{1}{2}, 1, \frac{1}{2}]$.

Lemma 22. *The prolongation can be written as* $R*_2^T : \mathbb{R}^{n_2} \to \mathbb{R}^{n_1}$, *for* $\mu^2 \in \mathbb{R}^{n_2}$, $(R*_2^T \mu^2) \in R^{n_1}$ *with*

$$(R *_2^T \mu^2)_{2i} = \mu_i^2 \quad (R *_2^T \mu^2)_{2i+1} = \frac{1}{2}(\mu_{i+1}^2 + \mu_i^2).$$

Noting that $u^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1$, $\mu^1 = \{\mu_i^1\}_{i=1}^{n_1}$, we obtain

$$\mu = \mu^1 + R *_2^T \mu^2 \quad \text{and} \quad u = u^1 + u^2.$$

Next we show how to realize the Algorithm 7 in vector and convolution form.

Algorithm 8 A two grid method $\mu = 2G1(b;\mu^0;2,\nu_1,\nu_2)$

Given μ^0.

Step1: Set $b^1 = b, \mu^1 \leftarrow \mu^0$.
For $i = 1 : \nu_1$

$$\mu^1 \leftarrow \mu^1 - \eta_1(A_1 * \mu^1 - b^1).$$

end for

Step 2: Set $r^1 = b^1 - A_1 * \mu^1, r^2 = R *_2 r^1, \mu^2 \leftarrow 0$.
For $i = 1 : \nu_2$

$$\mu^2 \leftarrow \mu^2 - \eta_2(A_2 * \mu^2 - r^2). \tag{8.26}$$

end for

Step 3: Update: $\mu = \mu^1 + R *_2^T \mu^2$.

Algorithm 9 A two grid algorithm $\mu = 2G1(b;\mu^0;2,\nu_1,\nu_2)$

Set up

$$b^1 = b, \quad \mu^1 = \mu^0.$$

Step 1: Smoothing and restriction from fine to coarse level (nested)
for $i = 1 : \nu_1$ **do**

$$\mu^1 \leftarrow \mu^1 + S^1 * (b^1 - A_1 * \mu^1). \tag{8.27}$$

end for
Step 2: Form restricted residual and set initial guess:

$$\mu^2 \leftarrow 0, \quad b^2 \leftarrow R *_2 (b^1 - A_1 * \mu^1),$$

for $i = 1 : \nu_2$ **do**

$$\mu^2 \leftarrow \mu^2 + S^2 * (b^2 - A_2 * \mu^2). \tag{8.28}$$

end for
Step 3: Prolongation and restriction from coarse to fine level

$$\mu^1 \leftarrow \mu^1 + R *_2^\top \mu^2.$$

$$\mu \leftarrow \mu^1.$$

8.1.6 Multilevel coarse grid corrections and a multigrid method

To describe a multigrid algorithm, we first need to have a multiple level of grids, say $\mathcal{T}_\ell$ with $\ell = 1 : J$ and $\mathcal{T}_1 = \mathcal{T}_h$ being the finest mesh. There are many ways to obtain multiple level of grids and one simple definition of the grid points in $\mathcal{T}_\ell$ is as follows:

$$x_i^\ell = \frac{i}{2^{J+1-\ell}}, \quad i = 1, 2, \cdots, N_\ell, \ell = 1, 2, \cdots, J,$$

where $N_\ell = 2^{J+1-\ell} - 1$. Note that $\mathcal{T}_{\ell-1}$ can be viewed as being obtained by adding midpoints of the subintervals in $\mathcal{T}_\ell$. For each ℓ the set of above nodes will be denoted by $\mathcal{N}_\ell$.

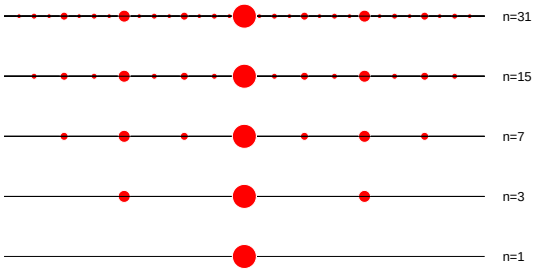


Fig. 8.8. Multiple grids in one dimension

With our previous experiences in two-grid method, the description of a multigrid method is not very difficult. In fact, the multiple level grids are treated by treating each two consecutive grids, say $\mathcal{T}_{\ell-1}$ versus $\mathcal{T}_\ell$. If we think $\mathcal{T}_{\ell-1}$ versus $\mathcal{T}_\ell$ like $\mathcal{T}_h$ and $\mathcal{T}_{2h}$, then there is not much new in the multigrid setting.

Let us give some details the definition of the restriction and prolongation matrices. The restriction matrix $R_{\ell-1}^\ell : R^{N_{\ell-1}} \mapsto R^{N_\ell}$ can be defined by

$$\gamma^\ell = R_{\ell-1}^\ell \gamma^{\ell-1} : \ \gamma_i^\ell = \frac{1}{2}\gamma_{2i-1}^{\ell-1} + \gamma_{2i}^{\ell-1} + \frac{1}{2}\gamma_{2i+1}^{\ell-1}. \tag{8.29}$$

The prolongation matrix $P_\ell^{\ell-1} : R^{N_\ell} \mapsto R^{N_{\ell-1}}$ can be defined as in (8.52) by

$$\epsilon^{\ell-1} = P_\ell^{\ell-1} \epsilon^\ell : \ \epsilon_{2i}^{\ell-1} = \epsilon_i^\ell, \epsilon_{2i+1}^{\ell-1} = \frac{1}{2}(\epsilon_i^\ell + \epsilon_{i+1}^\ell), \ i = 1 : N_\ell. \tag{8.30}$$

With the restriction and prolongation matrices in hands, we can now present a multilevel version of the earlier two-grid algorithm. As mentioned before, the idea is to repeat this two grid process for the coarse grid by using an even coarser grid. The resulting algorithm is just a desired multigrid algorithm. It is not hard to see that this kind of algorithm can be defined recursively or nonrecursively.

Using the convolution with stride notation, the restriction subprocess can also be written as $R*_2 : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_{\ell+1}}$, for any $v \in \mathbb{R}^{N_\ell}$, $u = (R *_2 v) \in R^{N_{\ell+1}}$ with

$$(R *_2 v)_i = \frac{1}{2} v_{2i-1} + v_{2i} + \frac{1}{2} v_{2i+1}$$

where $R = [\frac{1}{2}, 1, \frac{1}{2}]$.

Next let $u^{\ell+1} = \sum_{j=1}^{n_{\ell+1}} \mu_j^{\ell+1} \phi_j^{\ell+1} = (\mu^{\ell+1}, \phi^{\ell+1})_{l^2}$, then we have

$$\begin{aligned} u^{\ell+1} &= (\mu^{\ell+1}, \phi^{\ell+1})_{l^2} = (\mu^{\ell+1}, R *_2 \phi^\ell)_{l^2} = (R *_2^\top \mu^{\ell+1}, \phi^\ell)_{l^2} \\ &= \sum_{j=1}^{n_\ell} \left(R *_2^\top \mu^{\ell+1} \right)_j \phi_j^\ell. \end{aligned} \tag{8.31}$$

The prolongation subprocess can be written as $R*_2^T : \mathbb{R}^{N_{\ell+1}} \to \mathbb{R}^{N_\ell}$, for any $v \in \mathbb{R}^{N_{\ell+1}}$, $u = (R *_2^T v) \in R^{N_\ell}$ with

$$(R *_2^T v)_{2i} = v_i, \quad (R *_2^T v)_{2i+1} = \frac{1}{2}(v_{i+1} + v_i)$$

where $R = [\frac{1}{2}, 1, \frac{1}{2}]$.

Using the convolution notation, the subprocess to apply A_ℓ to a vector $v \in \mathbb{R}^{N_\ell}$ can be written as $A_\ell* : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_\ell}$, for any $v \in \mathbb{R}^{N_\ell}$, $r = (A_\ell * v) \in R^{N_\ell}$ with

$$(A_\ell * v)_i = \frac{1}{h_\ell}(-v_{i-1} + 2v_i - v_{i+1})$$

where $A = \frac{1}{h_\ell}[-1, 2, -1]$.

Algorithm 10 A multigrid algorithm $\mu = \text{MG1}(b; \mu^0; J, \nu_1, \cdots, \nu_J)$

Set up

$$b^1 = b, \quad \mu^1 = \mu^0.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
for $i = 1 : \nu_\ell$ **do**

$$\mu^\ell \leftarrow \mu^\ell + S^\ell * (b^\ell - A_\ell * \mu^\ell). \tag{8.32}$$

end for
Form restricted residual and set initial guess:

$$\mu^{\ell+1} \leftarrow 0, \quad b^{\ell+1} \leftarrow R *_2 (b^\ell - A_\ell * \mu^\ell), A_{\ell+1} = R *_2 A_\ell * (R*_2^\top).$$

end for
Prolongation and restriction from coarse to fine level
for $\ell = J - 1 : 1$ **do**

$$\mu^\ell \leftarrow \mu^\ell + R *_2^\top \mu^{\ell+1}.$$

end for

$$\mu \leftarrow \mu^1.$$

Next we consider another version with nonzero initial guess for the gradient descent method.

Step 1: Given $u^0 \in V_1$, apply ν_1-times gradient descent method for

$$\min_{v^1 \in V_1} J(v^1)$$

with initial guess u^0 to obtain $u^1 \in V_1$.

Let $u^0 = \sum_{i=1}^{n_1} \mu_i^0 \phi_i^1, \quad v^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1, \quad \mu^0 = \{\mu_i^0\}_{i=1}^{n_1}, \quad \mu^1 = \{\mu_i^0\}_{i=1}^{n_1}$.

Namely, $b^1 = b, \mu^1 = \mu^0$, for $i = 1 : \nu_1$

$$\mu^1 \leftarrow \mu^1 - \eta_1(A_1 * \mu^1 - b^1).$$

After ν_1 iterations, we obtain updated μ^1 and $u^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1$.

Step 2: Apply ν_2-times gradient descent iterations for

$$\min_{v^2 \in V_2} J(u^1 + v^2)$$

with initial guess $u^{2,0} = \Pi_1^2 u^1$ to obtain $u^2 \in V_2$, where $\Pi_1^2 u^1$ is the interpolation of u^1. Let

$$u^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1,\ v^2 = \sum_{i}^{n_2} \mu_i^2 \phi_i^2,\ u^{2,0} = \sum_{i}^{n_2} \mu_{2,i}^0 \phi_i^2,\ \mu^1 = \{\mu_i^1\}_{i=1}^{n_1},\ \mu^2 = \{\mu_i^2\}_{i=1}^{n_2},\ \mu_2^0 = \{\mu_{2,i}^0\}_{i=1}^{n_2}.$$

We have

$$\begin{aligned} J(u^1 + v^2) &= \frac{1}{2}\int_0^1 |(u^1 + v^2)'|^2 dx - \frac{1}{2}\int_0^1 f(u^1 + v^2) dx \\ &= J(u^1) + J(v^2) + \int_0^1 (u^1)'(v^2)' dx \\ &= \frac{1}{2}(\mu^1)^T A_1 * \mu^1 + \frac{1}{2}(\mu^2)^T A_2 * \mu^2 - (\mu^2)^T r^2 \end{aligned} \tag{8.33}$$

where

$$r_i^2 = \int_0^1 f\phi_i^2 - (u^1)'(\phi_i^2)' dx. \tag{8.34}$$

Now noting that

$$\phi_i^2 = \frac{1}{2}\phi_{2i-1}^1 + \phi_{2i}^1 + \frac{1}{2}\phi_{2i+1}^1, \tag{8.35}$$

Let $\phi^2 = \{\phi_i^2\}_{i=1}^{n_2}, \phi^1 = \{\phi_i^1\}_{i=1}^{n_1}$. Using the convolution with stride notation, we obtain

$$\phi^2 = R *_2 \phi^1 \tag{8.36}$$

with $R = [\frac{1}{2}, 1, \frac{1}{2}]$. Futhermore,

$$\begin{aligned} r_i^2 &= \langle f, \frac{1}{2}\phi_{2i-1}^1 + \phi_{2i}^1 + \frac{1}{2}\phi_{2i+1}^1\rangle - a(u^1, \frac{1}{2}\phi_{2i-1}^1 + \phi_{2i}^1 + \frac{1}{2}\phi_{2i+1}^1) \\ &= \frac{1}{2}b_{2i-1}^1 + b_{2i}^1 + \frac{1}{2}b_{2i+1}^1 - \Big(\frac{1}{2}(A_1 * \mu^1)_{2i-1} + (A_1 * \mu^1)_{2i} + \frac{1}{2}(A_1 * \mu^1)_{2i+1}\Big) \\ &= \frac{1}{2}(b^1 - A_1 * \mu^1)_{2i-1} + (b^1 - A_1 * \mu^1)_{2i} + \frac{1}{2}(b^1 - A_1 * \mu^1)_{2i+1} \\ &= \frac{1}{2}r_{2i-1}^1 + r_{2i}^1 + \frac{1}{2}r_{2i+1}^1, \end{aligned} \tag{8.37}$$

where $r^1 = b^1 - A_1 * \mu^1$. Using the convolution with stride notation, we have $r^2 = R *_2 r^1$ with $R = [\frac{1}{2}, 1, \frac{1}{2}]$.

Therefore applying gradient descent method ν_2-times for (8.34) reads:
$r^1 = b^1 - A_1 * \mu^1, r^2 = R *_2 r^1, \mu^2 = \mu^{2,0}$, for $i = 1 : \nu_2$

$$\mu^2 \leftarrow \mu^2 - \eta_2(A_2 * \mu^2 - r^2). \tag{8.38}$$

After ν_2 iterations, we obtain updated μ^2 and $u^2 = \sum_{i=1}^{n_2} \mu_i^2 \phi_i^2$.

Step 3: $u = u^1 + u^2 - u^{2,0}$.

Let $e^2 = u^2 - u^{2,0} = \sum_{i=1}^{n_2}(\mu_i^2 - \mu_i^{2,0})\phi_i^2 = \sum_{i=1}^{n_2} \epsilon_i^2 \phi_i^2, \epsilon_2 = \{\epsilon_i^2\}_{i=1}^{n_2}, \phi^2 = \{\phi_i^2\}_{i=1}^{n_2}$.

Therefore $e^2 = \sum_{i=1}^{n_2} \epsilon_i^2 \phi_i^2 = (\epsilon_2, \phi^2)_{l^2}$ and we have

$$\begin{aligned} e^2 &= (\epsilon_2, \phi^2)_{l^2} = (\epsilon_2, R *_1 \phi^1)_{l^2} = (R *_2^\top \epsilon_2, \phi^1)_{l^2} \\ &= \sum_{i=1}^{n_1} \Big(R *_2^\top \epsilon_2\Big)_i \phi_i^1 \end{aligned} \tag{8.39}$$

with $R = [\frac{1}{2}, 1, \frac{1}{2}]$. Hence prolongation can be written as $R *_2^T : \mathbb{R}^{n_2} \to \mathbb{R}^{n_1}$, for $\epsilon_2 \in \mathbb{R}^{n_2}, \epsilon_1 = (R *_2^T \epsilon_2) \in R^{n_1}$ with

$$(R *_2^T \epsilon_2)_{2i} = \epsilon_i^2, \quad (R *_2^T e^2)_{2i+1} = \frac{1}{2}(\epsilon_{2,i+1} + \epsilon_i^2).$$

Noting that $u^1 = \sum_{i=1}^{n_1} \mu_i^1 \phi_i^1$, $\mu^1 = \{\mu_i^1\}_{i=1}^{n_1}$, we obtain

$$\mu = \mu^1 + R *_2^T \epsilon_2 \quad \text{and} \quad u = u^1 + e^2 = u^1 + u^2 - u^{2,0}.$$

In summery, we also can write the two grid method with nonzero initial guess for gradient descent method in terms of FE functions as follows:

Next we show how to realize the Algorithm 11 in vector and convolution form.

Remark 9. We point out here that Algorithm 11 and Algorithmalg:2grid:vecno are independent of $u^{2,0}$ and μ_0^2.

Algorithm 11 A two grid method (in terms of FE functions)

Input u^0.

Step 1: Apply ν_1-times gradient descent iterations for

$$\min_{v^1 \in V_1} J(v^1)$$

with initial guess u^0 to obtain $u^1 \in V_1$.

Step 2: Apply ν_2-times gradient descent iterations for

$$\min_{v^2 \in V_2} J(u^1 + v^2)$$

with initial guess $u^{2,0} = \Pi_1^2 u^1$ to obtain $u^2 \in V_2$.

Step 3: Update: $u = u^1 + (u^2 - u^{2,0})$.

Algorithm 12 A two grid method (in terms of vectors and convolution form)

Given μ^0.

Step1: Set $b^1 = b, \mu^1 = \mu^0$. Apply ν_1-times gradient descent iterations:

For $i = 1 : \nu_1$

$$\mu^1 \leftarrow \mu^1 - \eta_1(A_1 * \mu^1 - b^1).$$

end for

Step 2: Set $r^1 = b^1 - A_1 * \mu^1, r^2 = R *_2 r^1 + A_2 * \mu_2^0, \mu^2 = \mu^{2,0}\mu^2 = \Pi_1^2 \mu^1$. Apply ν_2-times gradient descent iterations:

For $i = 1 : \nu_2$

$$\mu^2 \leftarrow \mu^2 - \eta_2(A_2 * \mu^2 - r^2). \tag{8.40}$$

end for

Step 3: Update: $\mu = \mu^1 + R *_2^T \epsilon_2$ with $\epsilon_2 = \mu^2 - \mu^{2,0}$.

Theorem 19. *The Algorithm 13 and Algorithm 12 are equivalent.*

Proof.

Introduce two versions of gradient descent method as follows:

In order to prove the theorem, it suffices to prove that $\mu^1 = \mu^2$. In fact, starting from Algorithm 13, noting that $r^0 = b - A * \mu^0$, we have

$$v^i + \mu^0 = v^{i-1} - \eta(A * v^{i-1} - r^0) + \mu^0 = v^{i-1} + \mu^0 - \eta(A * (v^{i-1} + \mu^0) - b) \quad \text{with} \quad v^0 = 0. \tag{8.41}$$

Denote $w^i = v^i + \mu^0$, then we can rewrite (8.41) as

$$w^i = w^{i-1} - \eta(A * w^{i-1} - b) \quad \text{with} \quad w^0 = \mu^0, \tag{8.42}$$

Algorithm 13 GD method with nonzero initial guess

Given μ^0, set $r^0 = b - A * \mu^0$, $v^0 = 0$.
For $i = 1 : \nu$

$$v^i = v^{i-1} - \eta(A * v^{i-1} - r^0).$$

end for

$$\mu^1 = \mu^0 + v^\nu.$$

Algorithm 14 GD method with nonzero initial guess

Given μ^0, set $w^0 = \mu^0$.
For $i = 1 : \nu$

$$w^i = w^{i-1} - \eta(A * w^{i-1} - b).$$

end for

$$\mu^2 = w^\nu.$$

which is exactly the w^i produced by Algorithm 14. Hence when $i = \nu$, we have $\mu^1 = \mu^0 + v^\nu = w^\nu = \mu^2$. □

Let us now give a numerical example to show how well the above two-grid method works.

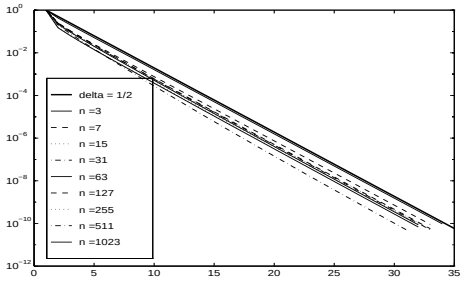


Fig. 8.9. Convergence history of the two level method for different number of unknowns (different values of h respectively). The thick line corresponds to damping factor $1/2$. Only one smoothing step is applied in the algorithm.

From the other hand,

$$\frac{d}{dt}J(u_h^\nu + tv_{2h}) = a(u_h^\nu + e_{2h}, v_{2h}) - \langle f, v_{2h}\rangle = 0,$$

namely

$$a(e_{2h}, v_{2h}) = \langle f, v_{2h}\rangle - a(u_h^\nu, v_{2h}) \tag{8.43}$$

Which means

$$A_{2h}\epsilon_{2h} = r^{2h},$$

where $r_i^{2h} = \langle f, \varphi_i^{2h}\rangle - a(u_h^\nu, \varphi_i^{2h})$.

In matrix form, let μ^ν denote the approximate solution obtained after ν-th iteration of the gradient descent method on the grid $\mathcal{T}_h$, and the residual is $\gamma^h = b - A_h\mu^\nu$. We consider the residual equation as follows:

$$A_h\epsilon^h = \gamma^h. \tag{8.44}$$

Here $\epsilon^h = \mu - \mu^\nu$. Since ϵ^h is smooth on $\mathcal{T}_h$, ϵ^h could be approximated well on a coarser grid. For simplicity, we assume that N_h is an odd integer and we take the coarse grid, denoted by $\mathcal{T}_{2h}$ to be a grid of size $2h$ with the following grid points

$$x_i^{2h} = x_{2i}, i = 1 : N_{2h} \equiv (N_h + 1)/2. \tag{8.45}$$

We would like to restrict the equation (8.44) to the above coarse grid and then solve the coarse grid equation. To do this, we first need to find a reasonable way for obtaining the restriction of (8.44) to the coarse grid. To this end, we can imagine that (8.44) is the linear system corresponding to a finite difference discretization of a two point boundary value problem:

$$-e'' = g, \quad x \in (0,1) \qquad e(0) = e(1) = 0. \tag{8.46}$$

Here g is some function satisfying

$$\gamma_i^h = hg(x_i), \tag{8.47}$$

and $A_h = (h)^{-1}\,\text{tridiag}(-1, 2, -1) \in R^{N_h \times N_h}$.

Now, we use finite difference to discretize (8.46) on the coarse grid $\mathcal{T}_{2h}$ and obtain the corresponding system as follows:

$$A_{2h}\epsilon^{2h} = \gamma^{2h}, \tag{8.48}$$

where $A_{2h} = (2h)^{-1}\,\text{tridiag}(-1, 2, -1) \in R^{N_{2h} \times N_{2h}}$ is the stiffness matrix from the coarse grid $\mathcal{T}_{2h}$ and

$$\gamma_i^{2h} = 2hg(x_i^{2h}). \tag{8.49}$$

Comparing (8.47) with (8.49) and using (8.45), we obtain the following relation for the components of the restricted vector γ^{2h}:

$$\gamma_i^{2h} = 2\gamma_{2i}^h. \tag{8.50}$$

The equation (8.48) with (8.50) is a desired restricted equation of (8.44) to the coarse grid $\mathcal{T}_{2h}$.

We would like to mention the following slightly different restriction is often used instead of (8.50):

$$\gamma_i^{2h} = \frac{1}{2}\gamma_{2i-1}^h + \gamma_{2i}^h + \frac{1}{2}\gamma_{2i+1}^h, \tag{8.51}$$

which is apparently very close to (8.50) since, for a smooth residual, $\gamma_{2i-1}^h \approx \gamma_{2i+1}^h \approx \gamma_{2i}^h$. We can imagine that using an averaged value should be better than a single value. We shall see in the next section, this is the most natural restriction in the finite element setting. This restriction also has a more interesting relation with the prolongation matrix that we will introduce in a moment.

The relation (8.50) or (8.51) defines a so-called restriction matrix $I_h^{2h} : R^{N_h} \mapsto R^{N_{2h}}$ such that

$$\gamma^{2h} = I_h^{2h}\gamma^h.$$

The equation (8.48) is half the size of (8.44). Let us solve this equation exactly and obtain:

$$\epsilon^{2h} = A_{2h}^{-1}\gamma^{2h}.$$

This ϵ^{2h} is the coarse grid correction of μ^ν. To add this correction to μ^ν, we need to think ϵ^{2h} to be the nodal value vector of a piecewise linear function defined on $\mathcal{T}_{2h}$. Apparently this piecewise linear function can be evaluated at the grid points of $\mathcal{T}_h$ as follows:

$$\tilde{\epsilon}_{2i}^h = \epsilon_i^{2h}, \tilde{\epsilon}_{2i-1}^h = \frac{1}{2}(\epsilon_i^{2h} + \epsilon_{i-1}^{2h}), \quad i = 1 : N_{2h}. \tag{8.52}$$

The above relation defines a so-called prolongation matrix $I_{2h}^h : R^{N_{2h}} \mapsto R^{N_h}$ such that

$$\tilde{\epsilon}^h = I_{2h}^h \epsilon^{2h}.$$

It is easy to see this prolongation matrix is related to the restriction matrix defined by (8.51) by the following identity:

$$I_{2h}^h = [I_h^{2h}]^T,$$

where T is the transpose. Because of this relationship, the restriction given by (8.51) is particularly interesting.

With this prolongation matrix, the approximation μ^ν can be updated as follows

$$\mu^\nu + I_{2h}^h \epsilon^{2h}.$$

Algorithm 15 A two grid method

Given μ^0, $\mu \leftarrow \mu^0$.

1. Fine grid smoothing: applying m times gradient descent iterations to obtain μ^ν.
2. Coarse grid correction: solving the residual equation restricted on the coarse grid $\mathcal{T}_{2h}$ to obtain
$$\epsilon^{2h} = A_{2h}^{-1}(I_h^{2h}\gamma^h).$$
3. Update: $\mu \leftarrow \mu^\nu + I_{2h}^h \epsilon^{2h}$.
4. Stop if converge or continue from step 1.

Apply restriction R_k^{k+1} to a vector $v \in \mathbb{R}^{N_k}$:
The subprocess to apply prolongation P_{k+1}^k to a vector $v \in \mathbb{R}^{N_k}$:
The subprocess to apply A_k to a vector $v \in \mathbb{R}^{N_k}$ is
We also can implement multi-grid method by recursion: Given a residual $\beta \in R^{N_k}$, and level k. To apply one step $(J - k + 1)$-level multigrid to solve

$$A_k\alpha = \beta \tag{8.53}$$

can be written as a function:

Algorithm 16 Apply Restriction to a vector

$u = \text{RESTRICTION}(v, k)$

for i **from** 1 **to** N_k

$$u(i) \leftarrow \frac{1}{2}v(2i-1) + v(i) + \frac{1}{2}v(2i+1)$$

endfor

return u.

Algorithm 17 Apply Prolongation to a vector

$u = \text{PROLONGATION}(v, k)$

$$u(1) \leftarrow \frac{1}{2}v(1)$$

for i **from** 1 **to** $N_k - 1$

$$u(2i) \leftarrow v(i)$$

$$u(2i+1) \leftarrow \frac{1}{2}(v(i+1) + v(i))$$

endfor

$$u(N_{k+1} - 1) \leftarrow v(N_k)$$

$$u(N_{k+1}) \leftarrow \frac{1}{2}v(N_k)$$

return u.

Algorithm 18 Apply Matrix

$r = \text{APPLYA}(v, k)$

$$r(1) = \frac{1}{h_k}(2v(1) - v(2))$$

for i **from** 2 **to** $N_k - 1$

$$r(i) \leftarrow \frac{1}{h_k}(2v(i) - v(i-1) - v(i+1))$$

endfor

$$r(N_k) = \frac{1}{h_k}(2v(N_k) - v(N_k - 1))$$

return r.

Algorithm 19 non-recursive operator form

$\alpha = \mathrm{MG}(\beta, k)$

$\sigma_k \leftarrow \dfrac{4}{h_k}$
if $k = J$
    $\alpha \leftarrow A_J^{-1}\beta$
else
    $\alpha \leftarrow \sigma_k^{-1}\beta,$
    $\beta \leftarrow \beta - A_k\alpha$
    $\alpha \leftarrow \alpha + P_{k+1}^k \mathrm{MG}(R_k^{k+1}\beta, k+1)$
endif
return α.

Algorithm 20 A recursive implementation

$\alpha = \mathrm{MG}(\beta, k)$

$\sigma_k \leftarrow \dfrac{4}{h_k}$
if $k = J$
    $\alpha \leftarrow A_J^{-1}\beta$
else
    $\alpha \leftarrow \sigma_k^{-1}\beta,$
    $\beta \leftarrow \beta - A_k\alpha$
    $\alpha \leftarrow \alpha + P_{k+1}^k \mathrm{MG}(R_k^{k+1}\beta, k+1)$
endif
return α.

8.2 1D and 2D Finite Element and Multigrid

Algorithm 21 Multigrid method

Given u^0.

1. For $\ell = 1 : J$

   Apply ν_ℓ times gradient descent iterations to an approximation e^{ν_ℓ} for $\min_{e \in V_\ell} J(u^{\ell-1} + e)$

   For $i = 1 : \nu_\ell$

   $$\mu \leftarrow \mu - \eta_2 (A_\ell * \mu - r^\ell)$$

   with

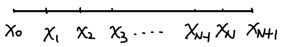
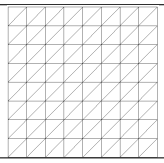
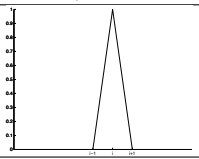
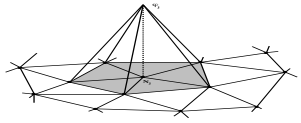
   $$r^\ell = R *_2 r^{\ell-1}.$$

   end for

   $$u^\ell = u^{\ell-1} + e^{\nu_\ell} \text{ with } e^{\nu_\ell} = \sum_{i=1}^{n_\ell} \mu_i \phi_i^\ell$$

   end for
2. Update: $u^0 \leftarrow u^J$.
3. Stop if converge or continue from step 1.

1D and 2D Comparison for Finite Element and Multigrid

$1D: \Omega = (0,1)$	$2D: \Omega = (0,1)\times(0,1)$
$\begin{cases} -u'' = f & x \in \Omega \\ u(0) = u(1) = 0 \end{cases}$	$\begin{cases} -\triangle u = f & x \in \Omega \\ u = 0 & x \in \partial\Omega \end{cases}$
$u = \arg\min_{v\in V} J(v)$	
$J(v) = \frac{1}{2}\int_0^1 \lvert v'\rvert^2 dx - \int_0^1 fv dx$	$J(v) = \frac{1}{2}\int_\Omega \lvert\nabla v\rvert^2 dx - \int_\Omega fv dx$
$V = \{v : \Omega \to \mathbb{R}^1$ is continuous and piecewise smooth, $v\lvert_{\partial\Omega} = 0\}$	
FE space: $V_h = \{v_h \in V, v_h$ is piecewise linear w.r.t $\mathcal{T}_h\}$	
	
$\phi_i(x)$	$\phi_{ij}(x,y)$
	
Find $u_h \in V_h$ s.t. $J(u_h) = \min_{v_h\in V_h} J(v_h)$	
$u_h = \sum_{i=1}^n \mu_i\phi_i(x)$	$u_h = \sum_{i,j=1}^n \mu_{ij}\phi_{ij}(x,y)$
Find $\mu \in \mathbb{R}^n$ s.t. $I(\mu) = \min_{\nu\in\mathbb{R}^n} I(\nu)$	Find $\mu \in \mathbb{R}^{n\times n}$ s.t. $I(\mu) = \min_{\nu\in\mathbb{R}^{n\times n}} I(\nu)$
$I(\nu) = \frac{1}{2}(A*\nu,\nu)_{l^2} - (b,\nu)_{l^2}$	
$A = \frac{1}{h}(-1,2,-1)$	$A = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$
$\mu = \arg\min I(\nu) \Longleftrightarrow \nabla J(\mu) = A*\mu - b = 0$	
GD Method: $\mu^{m+1} = \mu^m - \eta(A*\mu^m - b)$	
$\eta = \frac{h}{4}$	$\eta = \frac{1}{8}$

8.2.1 Multigrid algorithm for $A*\mu = f$

Algorithm 22 A multigrid algorithm $\mu = \mathrm{MG1}(f;\mu^0;J,\nu_1,\cdots,\nu_J)$

Set up

$$f^1 = f, \quad \mu^1 = \mu^0.$$

Smoothing and restriction from fine to coarse level (nested)

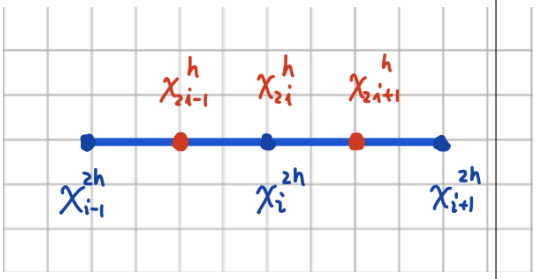
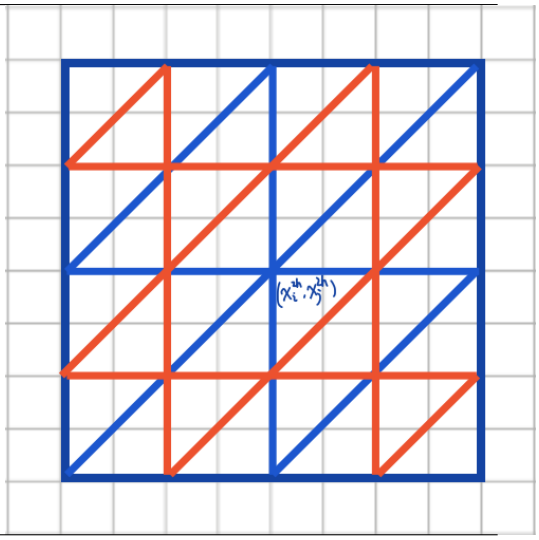
for $\ell = 1 : J$ **do**

for $i = 1 : \nu_\ell$ **do**

$$\mu^\ell \leftarrow \mu^\ell + S^\ell * (f^\ell - A_\ell * \mu^\ell). \tag{8.54}$$

end for

Basic multigrid components

	
$\phi_i^{2h} = \frac{1}{2}\phi_{2i-1}^h + \phi_{2i}^h + \frac{1}{2}\phi_{2i+1}^h$	$\phi_{i,j}^{2h} = \phi_{2i,2j}^h + \frac{1}{2}\left(\phi_{2i-1,2j-1}^h + \phi_{2i+1,2j+1}^h\right) + \frac{1}{2}\left(\phi_{2i-1,2j}^h + \phi_{2i,2j-1}^h + \phi_{2i+1,2j}^h + \phi_{2i,2j+1}^{2h}\right)$
$\Phi^{2h} = R *_2 \Phi^h$	
$R = (\frac{1}{2}, 1, \frac{1}{2})$	$R = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$

Form restricted residual and set initial guess:

$$\mu^{\ell+1} \leftarrow \Pi_\ell^{\ell+1}\mu^\ell, \quad f^{\ell+1} \leftarrow R *_2 (f^\ell - A_\ell * \mu^\ell) + A_{\ell+1} * \mu^{\ell+1},$$

end for
Prolongation and restriction from coarse to fine level
for $\ell = J - 1 : 1$ **do**

$$\mu^\ell \leftarrow \mu^\ell + R *_2^\top (\mu^{\ell+1} - \Pi_\ell^{\ell+1}\mu^\ell).$$

end for

$$\mu \leftarrow \mu^1.$$

Remark 10. The above multigrid method for the linear problem $A * \mu = b$ is independent of the choice of the interpolation operation $\Pi_\ell^{\ell+1} : \mathbb{R}^{n_\ell \times n_\ell} \mapsto \mathbb{R}^{n_{\ell+1} \times n_{\ell+1}}$ and in particular, we could take $\Pi_\ell^{\ell+1} := 0$. But such an operation is critical for nonlinear problems.

8.2.2 MgNet

Algorithm 23 $\mu^J = \text{MgNet1}(f; \mu^0; J, \nu_1, \cdots, \nu_J)$

Set up

$$f^1 = \theta * f, \quad \mu^1 = \mu^0.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
  for $i = 1 : \nu_\ell$ **do**

$$\mu^\ell \leftarrow \mu^\ell + \sigma \circ S^\ell * \sigma \circ (f^\ell - A_\ell * \mu^\ell). \tag{8.55}$$

  end for
  Form restricted residual and set initial guess:

$$\mu^{\ell+1} \leftarrow \Pi_\ell^{\ell+1} \mu^\ell, \quad f^{\ell+1} \leftarrow R *_2 (f^\ell - A_\ell * \mu^\ell) + A_{\ell+1} * \mu^{\ell+1},$$

end for

8.3 FEM and 2D in Convolution

Let us first briefly describe finite difference methods and finite element methods for the numerical solution of the following boundary value problem

$$-\Delta u = f, \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega, \quad \Omega = (0,1)^2. \tag{8.56}$$

For the x direction and the y direction, we consider the partition:

$$0 = x_0 < x_1 < \cdots < x_{n+1} = 1, \quad x_i = \frac{j}{n+1}, \quad (i = 0, \cdots, n+1); \tag{8.57}$$

$$0 = y_0 < y_1 < \cdots < y_{n+1} = 1, \quad y_j = \frac{j}{n+1}, \quad (j = 0, \cdots, n+1). \tag{8.58}$$

Such a uniform partition in the x and y directions leads us to a special example in two dimensions, a uniform square mesh $\mathbb{R}^2_h = \{(ih, jh); i, j \in \mathbb{Z}\}$ (Figure 8.3). Let $\Omega_h = \Omega \cap \mathbb{R}^2_h$, the set of interior mesh points and $\partial\Omega_h = \partial\Omega \cap \mathbb{R}^2_h$, the set of boundary mesh points.

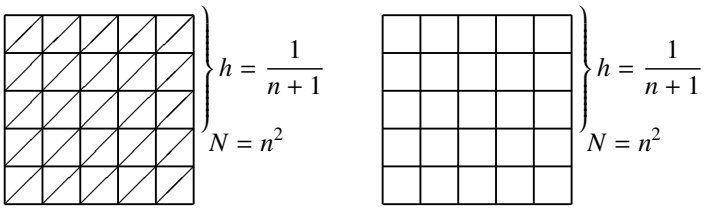


Fig. 8.10. Two-dimensional uniform grid for finite element and finite difference

8.4 Finite element methods

We consider two finite elements: continuous linear element and bilinear element. These two finite element methods find $u_h \in V_h$ such that

$$(\nabla u_h, \nabla v_h) = (f, v_h), \ \forall v_h \in V_h. \tag{8.59}$$

Basis functions ϕ_{ij} satisfy

$$\phi_{ij}(x_k, y_l) = \delta_{(i,j),(k,l)}. \tag{8.60}$$

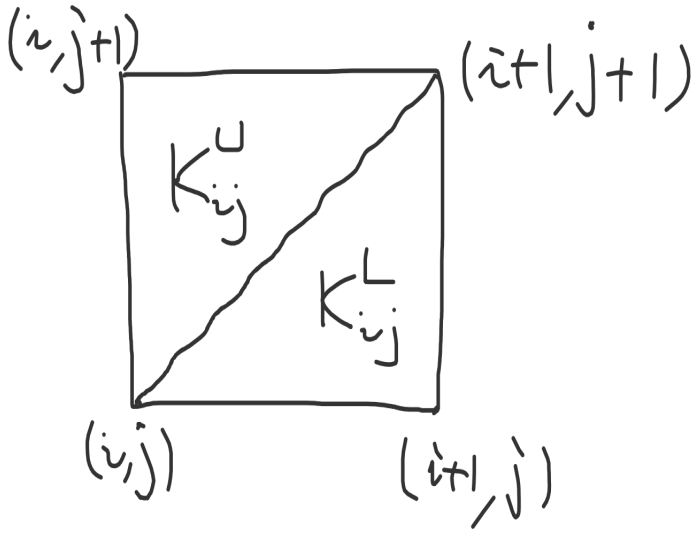


Fig. 8.11. The picture of $E_{i,j}$

8.4.1 Linear finite element

Continuous linear finite element discretization of (8.56) on the left triangulation in Fig 8.3. The discrete space for linear finite element is

$$\mathcal{V}_h = \{v_h : v_h|_K \in P_1(K) \text{ and } v_h \text{ is globally continuous}\}.$$

Denote $E_{i,j} = [x_i, x_{i+1}] \times [y_i, y_{i+1}] = K^U_{i,j} \cup K^L_{i,j}$. For linear element case,

(8.61)

$$\begin{aligned}
(\nabla \mathbf{u}_h, \nabla \mathbf{v}_h) &= \sum_{i,j=1}^{n} \int_{E_{i,j}} \nabla \mathbf{u}_h \cdot \nabla \mathbf{v}_h dxdy = \sum_{i,j=1}^{n} \int_{K^U_{i,j}} \nabla \mathbf{u}_h \cdot \nabla \mathbf{v}_h dxdy + \sum_{i,j=1}^{n} \int_{K^L_{i,j}} \nabla \mathbf{u}_h \cdot \nabla \mathbf{v}_h dxdy \\
&= \sum_{i,j=1}^{n} \int_{K^U_{i,j}} \Big(\frac{u_{i,j+1} - u_{i+1,j+1}}{h} \frac{v_{i,j+1} - v_{i+1,j+1}}{h} + \frac{u_{i,j+1} - u_{i,j}}{h} \frac{v_{i,j+1} - v_{i,j}}{h}\Big) dxdy \\
&+ \sum_{i,j=1}^{n} \int_{K^L_{i,j}} \Big(\frac{u_{i+1,j} - u_{i,j}}{h} \frac{v_{i+1,j} - v_{i,j}}{h} + \frac{u_{i+1,j} - u_{i+1,j+1}}{h} \frac{v_{i+1,j} - v_{i+1,j+1}}{h}\Big) dxdy \\
&= \sum_{i,j=1}^{n} \Big[(u_{i+1,j} - u_{i,j})(v_{i+1,j} - v_{i,j}) + (u_{i,j+1} - u_{i,j})(v_{i,j+1} - v_{i,j})\Big] \\
&= (A * u, v)_{l^2}
\end{aligned}$$

where $A = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ and $A * u$ is given by (8.62).

It is easy to verify that the formulation for the linear element method is

$$4u_{i,j} - (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) = f_{i,j}, \quad u_{i,j} = 0 \text{ if } i \text{ or } j \in \{0, n+1\}, \tag{8.62}$$

where

$$f_{i,j} = \int_\Omega f(x,y)\phi_{i,j}(x,y)\mathrm{d}x\mathrm{d}y \approx h^2 f(x_i, y_j). \tag{8.63}$$

Proposition 1. *The mapping $A*$ has following properties*

1. *A is symmetric, namely*

$$(A * u, v)_{l^2} = (u, A * v)_{l^2}.$$

2. $(A * v, v)_F > 0, \quad \text{if} \quad v \neq 0.$
3. *$A * u = f$ if and only if*

$$u \in \arg\min_{v \in \mathcal{V}_h} J(v) = \frac{1}{2}(A * v, v) - (f, v). \tag{8.64}$$

4. *The eigenvalues λ_{kl} and eigenvectors u^{kl} of A are given by*

$$\lambda_{kl} = 4(\sin^2 \frac{k\pi}{2(n+1)} + \sin^2 \frac{l\pi}{2(n+1)}),$$

$$u^{kl}_{ij} = \sin \frac{ki\pi}{n+1} \sin \frac{lj\pi}{n+1}, \ 1 \le i \le n, \ 1 \le j \le n,$$

and $\rho(A) < 8$. Furthermore,

$$\lambda_{n,n} = 8\cos^2 \frac{\pi}{2(n+1)} \approx 8(1 - (\frac{\pi}{2(n+1)})^2) \approx 8 - \frac{2\pi^2}{(n+1)^2}$$

8.4.2 Bilinear element

Continuous bilinear finite element discretization of (8.56) on the right mesh in Fig. 8.3. The discrete space for linear finite element is

$$\mathcal{V}_h = \{v_h : v_h|_K \in \{1, \ x, \ y, \ xy\} \text{ and } v_h \text{ is globally continuous}\}.$$

For bilinear element case, we have

(8.65)

$$
\begin{aligned}
(\nabla \mathbf{u}_h, \nabla \mathbf{v}_h) &= \sum_{i,j=1}^{n} \int_{E_{i,j}} \nabla \mathbf{u}_h, \nabla \mathbf{v}_h dxdy \\
&= \sum_{i,j=1}^{n} \int_{E_{i,j}} \left(\frac{(u_{i+1,j} - u_{i,j})(y_{j+1} - y)}{h^2} + \frac{(u_{i,j+1} - u_{i+1,j+1})(y - y_j)}{h^2} \right) \\
&\qquad \left(\frac{(v_{i+1,j} - v_{i,j})(y_{j+1} - y)}{h^2} + \frac{(v_{i,j+1} - v_{i+1,j+1})(y - y_j)}{h^2} \right) \\
&\qquad + \left(\frac{(u_{i,j+1} - u_{i,j})(x_{i+1} - x)}{h^2} + \frac{(u_{i+1,j} - u_{i+1,j+1})(x - x_i)}{h^2} \right) \\
&\qquad \left(\frac{(v_{i,j+1} - v_{i,j})(x_{i+1} - x)}{h^2} + \frac{(v_{i+1,j} - v_{i+1,j+1})(x - x_i)}{h^2} \right) dxdy \\
&= (A * u, v)_{l^2}.
\end{aligned}
$$

where $A = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$ and $A * u$ is given by (8.66).

And we have

$$8u_{ij} - (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + u_{i+1,j+1} + u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1}) = f_{i,j}, \tag{8.66}$$

and $u_{i,j} = 0$ if i or $j \in \{0, n+1\}$.

8.5 Piecewise (bi-)linear functions on multilevel grids

An image can be viewed as a function on a grid. Images with different resolutions can then be viewed as functions on grids of different sizes. The use of such multiple-grids is a main technique used in the standard multigrid method for solving discretized partial differential equations, and it can also be interpreted as a main ingredient used in convolutional neural networks (CNN) for image calssification.

An image can be viewed as a function on a grid [11] on a rectangle domain $\Omega \in \mathcal{R}^2$. Without loss of generality, we assume that the grid, $\mathcal{T}$, is of size

$$m = 2^s + 1 \quad n = 2^t + 1$$

for some integers $s, t \geq 1$. Starting from $\mathcal{T}_1 = \mathcal{T}$, we consider a sequence of coarse grids with $J = \min(s, t)$ (as depicted in Fig. 8.5 with $J = 4$):

$$\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_J \tag{8.67}$$

such that $\mathcal{T}_\ell$ consist of $m_\ell \times n_\ell$ grid points, with

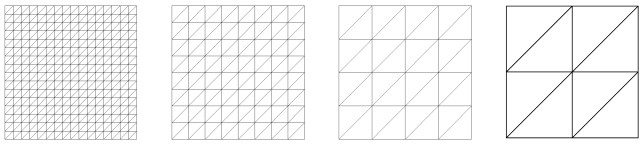


Fig. 8.12. multilevel grids for piecewise linear functions

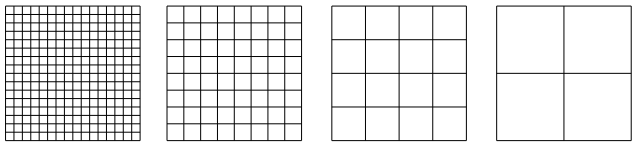


Fig. 8.13. multilevel grids for piecewise bilinear functions

$$m_\ell = 2^{s-\ell+1} + 1, \quad n_\ell = 2^{t-\ell+1} + 1. \tag{8.68}$$

The grid points of these grids can be given by

$$x_i^\ell = ih_\ell, y_j^\ell = jh_\ell, i = 0, \ldots, m_\ell - 1, j = 0, \ldots, n_\ell - 1.$$

Here $h_\ell = 2^{-s+\ell-1}a$ for some $a > 0$. The above geometric coordinates (x_i^ℓ, y_j^ℓ) are usually not used in image precess literatures, but they are relevant in the context of multigrid method for numerical solution of PDEs. We now consider piecewise bilinear (or linear) functions on the sequence of grids (9.1) and we obtain a nested sequence of linear vector spaces

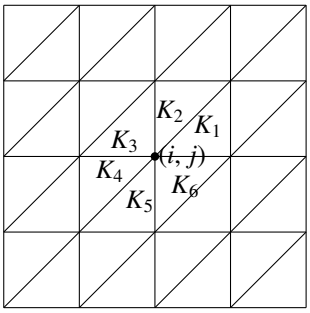
$$\mathcal{V}_1 \supset \mathcal{V}_2 \supset \ldots \supset \mathcal{V}_J. \tag{8.69}$$

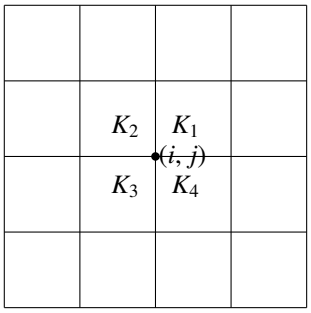
8.5.1 Nodal bases and dual bases on multilevel spaces

Here each $\mathcal{V}_\ell$ consists of all piecewise linear (or bilinear) functions with respect to the grid (9.1) and (9.2). Each $\mathcal{V}_\ell$ has a set of basis functions: $\phi_{i,j}^\ell \in \mathcal{V}_\ell$ satisfying:

$$\phi_{i,j}^\ell(x_p^\ell, y_q^\ell) = \delta_{(i,j),(p,q)} = \begin{cases} 1 & \text{if} \quad (p,q) = (i,j), \\ 0 & \text{if} \quad (p,q) \neq (i,j). \end{cases} \tag{8.70}$$

For the piecewise linear finite element space, the nodal basis function $\phi_{i,j}^\ell$ associate with each (x_i^ℓ, y_j^ℓ) (satisfying (8.70)) is given by





$$\phi^\ell_{i,j}(x,y) = \begin{cases} \frac{x^\ell_{i+1}-x}{h}, & (x,y)\in K_1, \\ \frac{y^\ell_{j+1}-y}{h}, & (x,y)\in K_2, \\ \frac{x-x^\ell_{i-1}-(y-y^\ell_j)}{h}, & (x,y)\in K_3, \\ \frac{x-x^\ell_{i-1}}{h}, & (x,y)\in K_4, \\ \frac{y-y^\ell_{j-1}}{h}, & (x,y)\in K_5, \\ \frac{x^\ell_{i+1}-x+y-y^\ell_j}{h}, & (x,y)\in K_6 \\ 0, & \text{elsewhere.} \end{cases} \tag{8.71}$$

shown in Fig. 8.14.

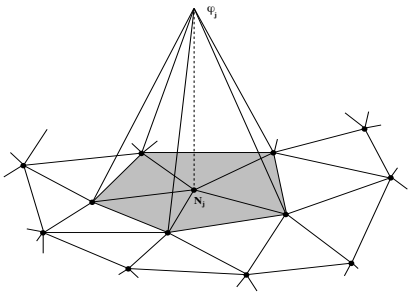


Fig. 8.14. Nodal basis for linear element.

For bilinear element, it is easy to see that the nodal basis function $\phi_{i,j}^{\ell}$ associated with each (x_i^{ℓ}, y_j^{ℓ}) (satisfying (8.70)) is given by

$$\phi_{i,j}^{\ell}(x,y) = \begin{cases} \frac{(x_{i+1}^{\ell}-x)(y_{j+1}^{\ell}-y)}{h^2}, & (x,y) \in K_1, \\ \frac{(x-x_{i-1}^{\ell})(y_{j+1}^{\ell}-y)}{h^2}, & (x,y) \in K_2, \\ \frac{(x-x_{i-1}^{\ell})(y-y_{j-1}^{\ell})}{h^2}, & (x,y) \in K_3, \\ \frac{(x_{i+1}^{\ell}-x)(y-y_{j-1}^{\ell})}{h^2}, & (x,y) \in K_4, \\ 0 & \text{elsewhere.} \end{cases} \tag{8.72}$$

Associated with the above nodal basis functions $\phi_{i,j}^{\ell}(x,y) \subset \mathcal{V}_{\ell}$, we define the corresponding dual basis functions $\psi_{i,j}^{\ell}(x,y) \subset \mathcal{V}_{\ell}$ satisfying

$$(\psi_{i,j}^{\ell}(x,y), \phi_{p,q}^{\ell}(x,y))_{L^2(\Omega)} = \delta_{(i,j),(p,q)}. \tag{8.73}$$

The existence of dual basis functions is obvious, but the exact expression of the dual basis functions are in general difficult to obtain. In fact, (8.73) is the only property that is needed in the application of dual basis.

We write $\mathbf{u}_h(x,y) = \sum_{i,j=1}^{n} u_{i,j}\phi_{i,j}(x,y)$, $\mathbf{v}_h(x,y) = \sum_{i,j=1}^{n} v_{i,j}\phi_{i,j}(x,y)$.

Lemma 23. *For bilinear functions, we have*

(8.74)

$$\begin{aligned} \phi_{i,j}^{\ell+1}(x,y) = \phi_{2i,2j}^{\ell}(x,y) &+ \frac{1}{2}\left(\phi_{2i-1,2j}^{\ell}(x,y) + \phi_{2i,2j-1}^{\ell}(x,y) + \phi_{2i+1,2j}^{\ell}(x,y) + \phi_{2i,2j+1}^{\ell}(x,y)\right) \\ &+ \frac{1}{4}\left(\phi_{2i-1,2j-1}^{\ell}(x,y) + \phi_{2i+1,2j-1}^{\ell}(x,y) + \phi_{2i+1,2j+1}^{\ell}(x,y) + \phi_{2i-1,2j+1}^{\ell}(x,y)\right). \end{aligned}$$

For linear functions, we have

$$\begin{aligned} \phi_{i,j}^{\ell+1}(x,y) = \phi_{2i,2j}^{\ell}(x,y) &+ \frac{1}{2}\left(\phi_{2i-1,2j-1}^{\ell}(x,y) + \phi_{2i+1,2j+1}^{\ell}(x,y)\right) \\ &+ \frac{1}{2}\left(\phi_{2i-1,2j}^{\ell}(x,y) + \phi_{2i,2j-1}^{\ell}(x,y) + \phi_{2i+1,2j}^{\ell}(x,y) + \phi_{2i,2j+1}^{\ell}(x,y)\right). \end{aligned} \tag{8.75}$$

Thus, for each $\mathbf{v}^{\boldsymbol{\ell}} \in \mathcal{V}_{\ell}, \mathbf{f}^{\boldsymbol{\ell}} \in \mathcal{V}_{\ell}' = \mathcal{V}_{\ell}$, we have

$$\mathbf{v}^{\boldsymbol{\ell}}(x,y) = \sum_{i=1}^{m_{\ell}}\sum_{j=1}^{n_{\ell}} v_{i,j}^{\ell}\phi_{i,j}^{\ell}(x,y), \quad \mathbf{f}^{\boldsymbol{\ell}}(x,y) = \sum_{i=1}^{m_{\ell}}\sum_{j=1}^{n_{\ell}} f_{i,j}^{\ell}\psi_{i,j}^{\ell}(x,y), \tag{8.76}$$

where

$$v_{i,j}^{\ell} = \mathbf{v}^{\ell}(x_i^{\ell}, y_j^{\ell}), \quad f_{i,j}^{\ell} = (\mathbf{f}^{\ell}, \phi_{i,j}^{\ell})_{L^2(\Omega)}. \tag{8.77}$$

Let us introduce the following tensors:

$$v^{\ell} = (v_{i,j}^{\ell}), \quad f^{\ell} = (f_{i,j}^{\ell}), \quad \phi^{\ell} = (\phi_{i,j}^{\ell}), \quad \psi^{\ell} = (\psi_{i,j}^{\ell}). \tag{8.78}$$

The following identities obviously hold:

$$\mathbf{v}^{\ell} = (v^{\ell}, \phi^{\ell})_{l^2}, \quad \mathbf{f}^{\ell} = (f^{\ell}, \psi^{\ell})_{l^2}, \quad (\mathbf{f}^{\ell}, \mathbf{v}^{\ell})_{L^2(\Omega)} = (f^{\ell}, v^{\ell})_{l^2}.$$

Denote $\phi^{\ell} = (\phi_{i,j}^{\ell}) \in \mathbb{R}^{m_\ell \times n_\ell}$, by the definitions of convolution (9.17) and stride (9.10), (8.74) means that

$$\phi^{\ell+1} = R *_2 \phi^{\ell},$$

where

$$R = \begin{cases} \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix} & \text{for bilinear functions;} \\ \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} & \text{for linear functions.} \end{cases} \tag{8.79}$$

8.6 Deconvolution

For any linear mapping $C : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{m'\times n'}$, its transpose is the unique linear mapping $C^{\top} : \mathbb{R}^{m'\times n'} \mapsto \mathbb{R}^{m\times n}$ satisfying

$$(C^{\top}u, v)_{l^2} = (u, Cv)_{l^2} \quad \forall\, u \in \mathbb{R}^{m'\times n'}, v \in \mathbb{R}^{m\times n}$$

Associated with any kernel K, a deconvolution is defined as the transpose of convolution with stride 2 with respect to the l^2-inner product as:

$$(u, K *_2^{\top} v)_{l^2} = (K *_2 u, v)_{l^2}, \tag{8.80}$$

with

$$u \in \mathbb{R}^{m\times n} \quad \text{and} \quad v \in \mathbb{R}^{\frac{m+1}{2}\times\frac{m+1}{2}}. \tag{8.81}$$

Lemma 24. *For any $K \in \mathbb{R}^{(2k+1)\times(2k+1)}$,*

$$K*_2^{\top} = \tilde{K} * \mathcal{S}^{\top}, \tag{8.82}$$

where $\tilde{K}$ is defined as

$$\tilde{K}_{p,q} = K_{-p,-q}, \quad p, q = -k : k. \tag{8.83}$$

Intuitively, if we take $K_{0,0}$ as the center for the convolutional kernel K, then $\tilde{K}$ is the central symmetry of K. In 2D case, it can also be understood as the rotation of π with respect to the center $K_{0,0}$.

Recalling the definition of deconvolution in (8.80), we have

$$\begin{aligned}(u, K *_2^\top v)_{l^2} &= (K *_2 u, v)_{l^2} = (\mathcal{S}C_K u, v)_{l^2} \\ &= (u, C_K^\top \mathcal{S}^\top v)_{l^2},\end{aligned} \tag{8.84}$$

with definition

$$\mathcal{S}^\top : \mathbb{R}^{\frac{m+1}{2}\times\frac{n+1}{2}} \mapsto \mathbb{R}^{m\times n}, \tag{8.85}$$

and

$$[\mathcal{S}^\top(f)]_{i,j} = \begin{cases} 0 & \text{if i or j is even,} \\ f_{i/2,j/2}, & \text{else.} \end{cases} \tag{8.86}$$

Thus to say, we have the simple version of the deconvolution for $K*$ as

$$K *_2^\top v = C_K^\top \circ \mathcal{S}^\top(v) = C_{\tilde{K}} \circ \mathcal{S}^\top(v) = \tilde{K} * \mathcal{S}^\top(v), \tag{8.87}$$

thus to say

$$K*_2^\top = \tilde{K} * \mathcal{S}^\top. \tag{8.88}$$

In short, we have the next decomposition

- convolution with stride = stride ∘ convolution,
- deconvolution with stride = transposed convolution ∘ transposed stride = convolution with the central symmetry of original kernel ∘ transposed stride.

Theorem 20. *Let us consider*

$$K = (K_{p,q}), \quad p, q = -1, 0, 1. \tag{8.89}$$

Then we have

$$K *_2^\top v = \tilde{K} * \mathcal{S}^\top(v).$$

As in (8.86) and the Lemma 24, we have the final version is

$$[K *_2^\top v]_{2i,2j} = K_{0,0} v_{i,j}, \tag{8.90}$$

with

$$[K *_2^\top v]_{2i-1,2j} = K_{0,1} v_{i-1,j} + K_{0,-1} v_{i,j}, \quad [K *_2^\top v]_{2i,2j-1} = K_{1,0} v_{i,j} + K_{-1,0} v_{i,j-1}, \tag{8.91}$$

and

$$[K *_2^\top v]_{2i-1,2j-1} = K_{1,1} v_{i,j} + K_{-1,1} v_{i-1,j} + K_{1,-1} v_{i,j-1} + K_{-1,-1} v_{i-1,j-1}. \tag{8.92}$$

Remark 11. Deconvolution can obviously be also defined for general stride s, but we believe it is sufficient to use $s = 2$ in most applications.

8.7 Linear feature mappings

We consider the following linear mapping

$$\mathbf{Au} = \mathbf{f} \tag{8.93}$$

where

$$\mathbf{A} : \mathcal{V} \mapsto \mathcal{V}'. \tag{8.94}$$

We consider the restriction of the mapping $\mathbf{A}_1 \equiv \mathbf{A}$ on the coarser multilevel spaces:

$$\mathbf{A}_\ell : \mathcal{V}_\ell \mapsto \mathcal{V}'_\ell \tag{8.95}$$

and the corresponding equation read as:

$$\mathbf{A}_\ell \mathbf{u}^\ell = \mathbf{f}^\ell. \tag{8.96}$$

In image process, we can view $\mathbf{f}$ as the input images and $\mathbf{u}$ as the extracted features of the original image $\mathbf{f}$. We then view $\mathbf{f}^\ell$ as the projection of images on a coarser resolution and $\mathbf{u}^\ell$ as the extracted features of the coarsened image $\mathbf{f}^\ell$.

One main question is how to obtain coarser images and features defined by (8.96) from the original equation (8.93). We now consider a special technique.

We define $\mathbf{u}^\ell \in \mathcal{V}_\ell$ by

$$(\mathbf{Au}^\ell, \mathbf{v}^\ell) = (\mathbf{f}, \mathbf{v}^\ell), \quad \forall \mathbf{v}^\ell \in \mathcal{V}_\ell \tag{8.97}$$

Lemma 25. *The restricted $\mathbf{u}^\ell \in \mathcal{V}_\ell$ defined by (8.97) satisfies (8.96) if $\mathbf{A}_\ell : \mathcal{V}_\ell \mapsto \mathcal{V}'_\ell$ and $\mathbf{f}_\ell \in \mathcal{V}'_\ell$ are defined by*

$$(\mathbf{A}_\ell \mathbf{u}^\ell, \mathbf{v}^\ell) = (\mathbf{Au}^\ell, \mathbf{v}^\ell), \quad \forall \mathbf{v}^\ell \in \mathcal{V}_\ell \tag{8.98}$$

$$(\mathbf{f}^\ell, \mathbf{v}^\ell) = (\mathbf{f}, \mathbf{v}^\ell), \quad \forall \mathbf{v}^\ell \in \mathcal{V}_\ell \tag{8.99}$$

8.7.1 Restriction and prolongation under the convolution notation

Now we derive the restriction and prolongation as follows. We show the details for the case of bilinear functions here. The case of linear function can be shown similarly. Let $f^{\ell+1}_{i,j} = (\mathbf{f}^\ell, \phi^{\ell+1}_{i,j})_{L^2(\Omega)}$, then we have

(8.100)

$$\begin{aligned} f^{\ell+1} &= \int_\Omega \mathbf{f}^\ell \phi^{\ell+1} = \sum_{i=1}^{m_\ell} \sum_{j=1}^{n_\ell} \int_\Omega f^\ell_{i,j} \psi^\ell_{i,j} \left[(R *_2) \phi^\ell \right] = \sum_{i=1}^{m_\ell} \sum_{j=1}^{n_\ell} f^\ell_{i,j} (R *_2) \int_\Omega \psi^\ell_{i,j} \phi^\ell \\ &= \sum_{i=1}^{m_\ell} \sum_{j=1}^{n_\ell} (R *_2) f^\ell_{i,j} e_i e_j^T = R *_2 f^\ell. \end{aligned}$$

Hence the restriction

$$R_\ell^{\ell+1} : \mathbb{R}^{m_\ell \times n_\ell} \mapsto \mathbb{R}^{m_{\ell+1} \times n_{\ell+1}}$$

is obtain by $R_\ell^{\ell+1} f^\ell = R *_2 f^\ell$ with $R \in \mathbb{R}^{3\times 3}$ given by (8.79), namely

$$\begin{aligned} f_{i,j}^{\ell+1} &= f_{2i,2j}^\ell + \frac{1}{2}(f_{2i-1,2j}^\ell + f_{2i,2j-1}^\ell + f_{2i+1,2j}^\ell + f_{2i,2j+1}^\ell) \\ &\quad + \frac{1}{4}\left(f_{2i-1,2j-1}^\ell + f_{2i+1,2j-1}^\ell + f_{2i+1,2j+1}^\ell + f_{2i-1,2j+1}^\ell\right). \end{aligned} \tag{8.101}$$

Next let $\mathbf{u}^{\ell+1} = \sum_{i=1}^{m_{\ell+1}} \sum_{j=1}^{n_{\ell+1}} u_{i,j}^{\ell+1} \phi_{i,j}^{\ell+1} = (\mathbf{u}^{\ell+1}, \phi^{\ell+1})_{l^2}$, then we have

$$\begin{aligned} \mathbf{u}^{\ell+1} &= (u^{\ell+1}, \phi^{\ell+1})_{l^2} = (u^{\ell+1}, R *_2 \phi^\ell)_{l^2} = (R *_2^\top u^{\ell+1}, \phi^\ell)_{l^2} \\ &= \sum_{i=1}^{m_\ell} \sum_{j=1}^{n_\ell} \left(R *_2^\top u^{\ell+1}\right)_{i,j} \phi_{i,j}^\ell. \end{aligned} \tag{8.102}$$

Namely

$$\mathbf{u}^{\ell+1}(x_i^\ell, y_j^\ell) = \left(R *_2^\top u^{\ell+1}\right)_{i,j}.$$

And we obtain the prolongation

$$P_{\ell+1}^\ell = R *_2^\top : \mathbb{R}^{m_{\ell+1} \times n_{\ell+1}} \mapsto \mathbb{R}^{m_\ell \times n_\ell}$$

is defined by

$$u_{2i,2j}^\ell = u_{i,j}^{\ell+1},$$

$$u_{2i-1,2j}^\ell = \frac{1}{2}(u_{i,j}^\ell + u_{i-1,j}^\ell), \quad u_{2i,2j-1}^\ell = \frac{1}{2}(u_{i,j}^{\ell+1} + u_{i,j-1}^{\ell+1})$$

and

$$u_{2i-1,2j-1}^\ell = \frac{1}{4}(u_{i,j}^{\ell+1} + u_{i-1,j}^{\ell+1} + u_{i-1,j-1}^{\ell+1} + u_{i,j-1}^{\ell+1}).$$

In summery, we have the restriction and prolongation as follows:

Lemma 26. *The restriction*

$$R_\ell^{\ell+1} : \mathbb{R}^{m_\ell \times n_\ell} \mapsto \mathbb{R}^{m_{\ell+1} \times n_{\ell+1}} \ \textit{is} \ R_\ell^{\ell+1} = R *_2$$

and the prolongation

$$P_{\ell+1}^\ell : \mathbb{R}^{m_{\ell+1} \times n_{\ell+1}} \mapsto \mathbb{R}^{m_\ell \times n_\ell} \ \textit{is} \ P_{\ell+1}^\ell = R *_2^\top$$

where

$$R = \begin{cases} \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix} & \textit{for bilinear functions}; \\ \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} & \textit{for linear functions}. \end{cases} \tag{8.103}$$

Let $(\phi^\ell_{i,j})$ is a basis of $\mathcal{V}_\ell$, for any $\mathbf{u}^\ell \in \mathcal{V}_\ell$, then $\mathbf{u}^\ell = \sum_{i=1}^{m_\ell}\sum_{j=1}^{n_\ell} u^\ell_{i,j}\phi^\ell_{i,j}$, and we denote $u^\ell = (u^\ell_{i,j}) \in \mathbb{R}^{m_\ell \times n_\ell}$ the matrix representation of $\mathbf{u}^\ell$ under the basis $(\phi^\ell_{i,j})$. Let $(\psi^\ell_{s,t})$ is a basis of $\mathcal{V}'_\ell$ which is dual to $(\phi^\ell_{i,j})$. Denote $A_\ell = (a^\ell_{stij})$ the tensor representation of $\mathbf{A}_\ell : \mathcal{V}_\ell \mapsto \mathcal{V}'_\ell$ and defined as

$$\mathbf{A}_\ell \phi^\ell_{i,j} = \sum_{s=1}^{m_\ell}\sum_{t=1}^{n_\ell} a^\ell_{stij}\psi^\ell_{s,t}.$$

Hence $a^\ell_{stij} = (\mathbf{A}_\ell\phi^\ell_{i,j}, \phi^\ell_{s,t})$. From $(\mathbf{A}_{\ell+1}\phi^{\ell+1}_{i,j}, \phi^{\ell+1}_{s,t}) = (\mathbf{A}_\ell\phi^{\ell+1}_{i,j}, \phi^{\ell+1}_{s,t})$, we have

$$a^{\ell+1}_{stij} = \sum_{r=1}^{m_\ell}\sum_{q=1}^{n_\ell}\left(\sum_{k=1}^{m_\ell}\sum_{m=1}^{n_\ell} P^{\ell,\ell+1}_{kmij} a^\ell_{rqkm}\right) P^{\ell,\ell+1}_{rqst}$$

Where $P^{\ell,\ell+1} = (P^{\ell,\ell+1}_{rqst})$ is the tensor representation of the prolongation $P^\ell_{\ell+1}$.

Consider the finite element method on two different grids $\mathcal{T}_\ell$, $\mathcal{T}_{\ell+1}$, $h_{\ell+1} = 2h_\ell, \mathcal{V}_{\ell+1} \subset \mathcal{V}_\ell$. With the restriction $R^{\ell+1}_\ell$ and prolongation $P^\ell_{\ell+1}$ obtained in Lemma 26, we have the following relationship to define coarse operation

$$\begin{aligned} A_{\ell+1} &= R^{\ell+1}_\ell A_\ell P^\ell_{\ell+1}. \\ &= R *_2 A_\ell * (R*_2^\top), \quad (\ell = 1 : J-1), \end{aligned} \tag{8.104}$$

with $A_1 = A$.

Theorem 21. *If R is consistent with A_ℓ which means that R should be linear or bi-linear as A_ℓ, then we have the $A_{\ell+1}$ operation in coarse grid defined in (8.104) is the same with A_ℓ.*

Proof. For any $u_{\ell+1}$ and $v_{\ell+1}$ in $\mathcal{V}_{\ell+1}$, it remains to prove that

$$(A_\ell P^\ell_{\ell+1}u_{\ell+1}, P^\ell_{\ell+1}v_{\ell+1}) = (A_{\ell+1}u_{\ell+1}, v_{\ell+1})$$

where A_ℓ and $A_{\ell+1}$ are the tensor representation of $\mathbf{A}_\ell$ and $\mathbf{A}_{\ell+1}$.

We can also view them as convolutions. By the definition of operators $R^{\ell+1}_\ell$ and $P^\ell_{\ell+1}$, a direct computation gives the above result. □

Proof. By the definition above, we have that

$$A_{\ell+1}(v) = \mathcal{S}\Big((R * A_\ell * R) * \mathcal{S}^\top(v)\Big), \tag{8.105}$$

because of the properties of convolution we know that

$$(R * A_\ell * R)* = K*, \tag{8.106}$$

for some

$$K \in \mathbb{R}^{7\times 7}.$$

Then we have the next computation for $A_{\ell+1}(v)$

$$
\begin{aligned}
[A_{\ell+1}(v)]_{i,j} &= [\mathcal{S}\Big((R * A_\ell * R) * \mathcal{S}^\top(v)\Big)]_{i,j}, \\
&= [K * \mathcal{S}^\top(v)]_{2i,2j}, \\
&= \sum_{p,q=-3}^{3} [\mathcal{S}^\top(v)]_{2i+p,2j+q} K_{p,q}, \\
&= \sum_{p,q=-1}^{1} [\mathcal{S}^\top(v)]_{2(i+p),2(j+q)} K_{2p,2q}, \\
&= \sum_{p,q=-1}^{1} v_{i+p,j+q} \hat{K}_{p,q},
\end{aligned}
\tag{8.107}
$$

Thus to say, we have

$$
A_{\ell+1}(v) = \hat{K} * v, \tag{8.108}
$$

with

$$
\hat{K}_{p,q} = K_{2p,2q}, \quad p, q = -1, 0, 1,
$$

with K is defined in (8.106).

Then by the direct computation of (8.106) as

$$
(R * A_\ell * R)* = K*
$$

and take the even index we have that

$$
A_{\ell+1} = \hat{K} = A_\ell, \tag{8.109}
$$

if R is consistent with A_ℓ which means that R should be linear or bi-linear as A_ℓ. □

8.8 Multigrid for finite element methods

By the definition of convolution (9.17), we can rewrite (8.62) and (8.66) as follows:

(8.110)

$$
A* : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}, \quad A*u = f, \quad \Leftrightarrow \quad u = \arg\min J(v) = \arg\min \left(\frac{1}{2}(A * v, v) - (f, v)_{l^2} \right)
$$

where $u = (u_{ij})$, $f = (f_{ij})$,

$$A = \begin{cases} \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} & \text{for linear finite element,} \\ \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} & \text{for bilinear finite element.} \end{cases} \tag{8.111}$$

It is easy to see that

$$\nabla J(v) = A * v - f := -r, \quad r = f - A * v.$$

Applying the gradient descent method to (8.64), we obtain the following iterative method:

$$u^{k+1} = u^k + \eta r^k, \quad r^k = f - A * u^k. \tag{8.112}$$

Here we can clearly see that gradient descent method is equivalent to damped Jacobi method. Usually, we call this as smoother.

Lemma 27. *The gradient descent method (8.112) for linear finite element converges if $\eta = \frac{1}{8}$, and the one for bilinear finite element converges if $\eta = \frac{1}{16}$. Furthermore, the high frequence in $u - u^k$ are damped very rapidly.*

Proof. According to (8.112) ,

$$u^{k+1} - u = (I - \eta A) * (u^k - u).$$

The gradient descent method (8.112) converges if $\rho((I - \eta A)*) < 1$, namely $\eta\rho(A*) < 2$. For linear finite element, if $\eta = \frac{1}{8}$, $\rho(A*) < 8$, thus the gradient descent method (8.112) converges. For bilinear finite element, if $\eta = \frac{1}{16}$, $\rho(A*) < 16$, thus the gradient descent method (8.112) converges.

For linear finite element, let (λ_i, v_i) satisfy $A * v_i = \lambda_i v_i$ and $0 < \lambda_1 \leq \lambda_2 \leq \cdots \lambda_N$ with $N = n^2$. Expand the error $u^k - u$ in terms of eigenvectors v_i, namely,

$$u^k - u = \sum_{i=1}^{N} a_i^k v_i.$$

Then

$$u^k - u = \sum_{i=1}^{N} a_i^k (1 - \eta\lambda_i) v_i = \sum_{i=1}^{N} a_i^0 (1 - \eta\lambda_i)^k v_i.$$

According to Proposition 1, it is easy to see that

$$1 - \frac{1}{8}\lambda_N \approx \frac{\pi^2}{4(n+1)^2} \ll 1.$$

For $\eta = \frac{1}{8}$, the coefficient $a_N^0(1 - \frac{1}{8}\lambda_N)^k$ of v_N approximates to zero much faster. This means that high frequency in the error will damp rapidly. □

For an initial guess u^0, the left picture in Fig 8.15 plots the error $u - u^0$ and the the right one plots the error $u - u^1$. Fig 8.15 shows that the high frequency in the error of the initial guess u^0 is damped after one step of smoothing and results in a smoother error $u - u^1$. Next, we make the following particular choice:

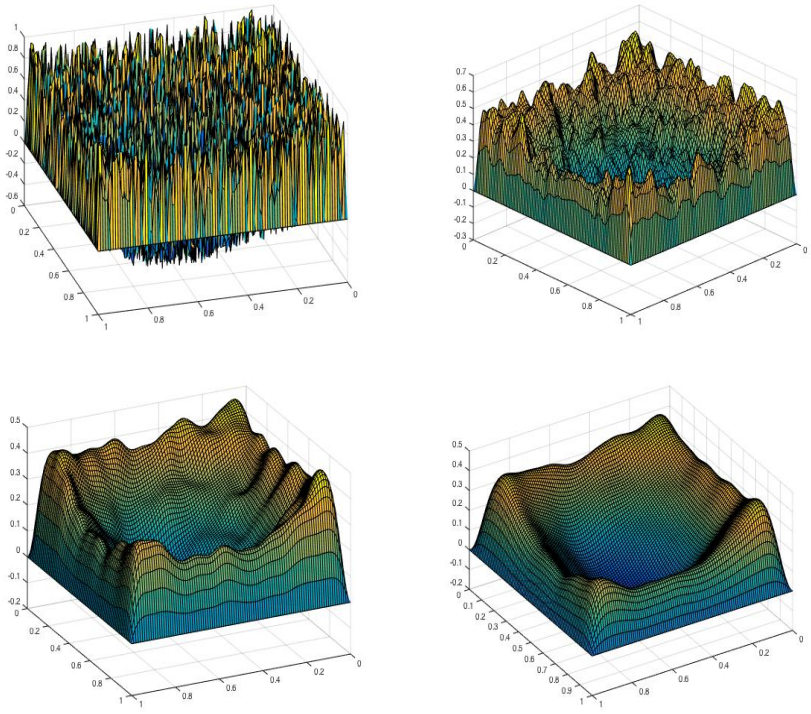


Fig. 8.15. The errors of an random initial guess u^0, u^{10}, u^{50} and u^{100}.

$$\eta = \frac{1}{8}. \tag{8.113}$$

The gradient descent method can be written in terms of $S_0 : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{m\times n}$ satisfying

$$u^1 = (S_0 f) = \frac{1}{8} f, \tag{8.114}$$

for equation (8.62) with initial guess zero. If we apply this method twice, then

$$u^2 = S_1(f) = S_0 f + S_0(f - A * (S_0 f)),$$

with element-wise form

$$u^2_{i,j} = \frac{3}{16} f_{i,j} + \frac{1}{64}(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}). \tag{8.115}$$

Then by the definition of convolution (9.17), we have

$$u^1 = S_0 * f \quad u^2 = S_1 * f. \tag{8.116}$$

with

$$S_0 = \frac{1}{8}, \tag{8.117}$$

and

$$S_1 = \frac{1}{64}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 12 & 1 \\ 0 & 1 & 0 \end{pmatrix}. \tag{8.118}$$

Next, we make the following particular choice for the bilinear case:

$$\eta = \frac{1}{16}. \tag{8.119}$$

The gradient descent method can be written in terms of $S_0 : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{m\times n}$ satisfying

$$u^1 = (S_0 f)_{i,j} = \frac{1}{16} f_{i,j}, \tag{8.120}$$

for equation (8.66) with initial guess zero. If we apply this method twice, then

$$u^2 = S_1(f) = S_0 f + S_0(f - A * (S_0 f)),$$

with element-wise form

$$u^2_{i,j} = \frac{3}{32} f_{i,j} + \frac{1}{256}(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} + f_{i+1,j+1} + f_{i-1,j-1} + f_{i-1,j+1} + f_{i+1,j-1}). \tag{8.121}$$

Then by the definition of convolution (9.17), we have

$$u^1 = S_0 * f \quad u^2 = S_1 * f. \tag{8.122}$$

with

$$S_0 = \frac{1}{16}, \tag{8.123}$$

and

$$S_1 = \frac{1}{256}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 24 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \tag{8.124}$$

We note the gradient descent method (8.112) can be written as

$$u^k = u^{k-1} + S_0 * (f - A * u^{k-1}). \tag{8.125}$$

$$u^{2k} = u^{2(k-1)} + S_1 * (f - A * u^{2(k-1)}). \tag{8.126}$$

We can sometimes use S_1 and the above identity to define a smoother directly, namely

$$u^k = u^{k-1} + S_1 * (f - A * u^{k-1}). \tag{8.127}$$

Similarly, with the smoother obtained by (8.122), we can define $S^\ell : \mathbb{R}^{m_\ell \times n_\ell} \mapsto \mathbb{R}^{m_\ell \times n_\ell}$.

First solve the problem on the fine grid $\mathcal{T}_\ell$, denote the solution by u_ℓ, the error by $e = u - u_\ell$ and the residual

$$r^\ell = f - A_\ell * u_\ell.$$

It is obvious that $A_\ell * e_\ell = r^\ell$. We need to solve the residual equation

$$A_\ell * e_\ell = r^\ell. \tag{8.128}$$

The idea of multigrid method is to solve this residual equation on the coarse grid space $\mathcal{V}_{\ell+1}$ and repeat the process until coarsest grid.

We note that the restriction of (8.128) to the coarse level $\ell + 1$ is

$$A_{\ell+1} * e_{\ell+1} = r^{\ell+1}$$

with $r^{\ell+1} = R_\ell^{\ell+1} r^\ell = R * r^\ell$.

Denote the finite element approximation to e_ℓ on the coarse grid $\mathcal{T}_{\ell+1}$ by $e_{\ell+1}$. Interpolate the error $e_{\ell+1}$ back to the fine space $\mathcal{V}_\ell$ and add the resulting residual to $u_{\ell+1}$, that is

$$u_\ell \leftarrow u_\ell + P_{\ell+1}^\ell e_{\ell+1}$$

Now using the smoother S^ℓ, prolongation $P_{\ell+1}^\ell$, restriction $R_\ell^{\ell+1}$ and mapping A^ℓ as given in (8.104), we can formulate the following algorithm as a major component of a multigrid algorithm.

Algorithm 24 $(u^1, u^2, \cdots, u^J) = \text{MG0}(f; u^0; J, \nu_1, \cdots, \nu_J)$

Set up

$$f^1 = f, \quad u^1 = u^0.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
for $i = 1 : \nu_\ell$ **do**

$$u^\ell \leftarrow u^\ell + S^\ell * (f^\ell - A_\ell * u^\ell). \tag{8.129}$$

end for
Form restricted residual and set initial guess:

$$u^{\ell+1,0} \leftarrow 0, \quad f^{\ell+1} \leftarrow R *_2 (f^\ell - A_\ell * u^\ell), A_{\ell+1} = R *_2 A_\ell * (R *_2^\top).$$

end for

Here S^ℓ can be chosen as S_0 or S_1 as definition in (8.123) and (8.124).

Using the above algorithm, there are different multigrid algorithms such as: \-cycle, V-cycle and W-cycle. Let us now only give one special form of multigrid algorithm for solving (8.56) as follows.

Algorithm 25 $u = \text{MG1}(f; u^0; J, \nu_1, \cdots, \nu_J)$

$$u \leftarrow u^0.$$

$$(u^1, u^2, \cdots, u^J) = \text{MG0}(f; u; J, \nu_1, \cdots, \nu_J).$$

Prolongation and restriction from coarse to fine level
for $\ell = J - 1 : 1$ **do**

$$u^\ell \leftarrow u^\ell + R *_2^\top u^{\ell+1}.$$

end for

$$u \leftarrow u^1.$$

If we add the post-smoothing with a symmetric form which means we use $[S^\ell *]^\top$ as the smoother, then we can get the V-cycle version multigrid algorithm.

Algorithm 26 $u = \text{MG2}(f; u^0; J, \nu_1, \cdots, \nu_J; \nu'_1, \cdots, \nu'_J)$

$$(u^1, u^2, \cdots, u^J) = \text{MG0}(f; u^0; J, \nu_1, \cdots, \nu_J).$$

Prolongation and restriction from coarse to fine level
for $\ell = J - 1 : 1$ **do**

$$u^\ell \leftarrow u^\ell + R *_2^\top u^{\ell+1}.$$

for $i = 1 : \nu'_\ell$ **do**

$$u^\ell \leftarrow u^\ell + \tilde{S}^\ell * (f^\ell - A_\ell * u^\ell)$$

end for
end for

$$u = u^1.$$

Here $\tilde{S}^\ell$ means the central symmetry of kernel for smoother S^ℓ as in the definition of (8.83) in Lemma 24.

We note that

$$\text{MG1}(f; u^0; J, \nu_1, \cdots, \nu_J) = \text{MG2}(f; u^0; J, \nu_1, \cdots, \nu_J; 0, \cdots, 0).$$

Either MG1 or MG2 only represents one cycle in a multigrid process. There many different ways to use this basis multigrid cycle. For a given iterate u, we need to define a metric to measure the accuracy of u. One way to define it is:

$$\text{error}(u) = \|f - A * u\|/\|f - A * u^0\|.$$

Sometimes, when we debug a code, we can first try to find the exact solution. u_{exact}, and then define

$$\text{error}(u) = \|u_{\text{exact}} - u\|_A.$$

Below is one example fo algorithm for application of the basic multigrid cycle, say MG1:

Algorithm 27 $u = \text{multigrid1}(f; u^0; J, \nu_1, \cdots, \nu_J; \text{tol})$;

$$u \leftarrow u^0.$$

while $\text{error}(u) \geq tol$ **do**

$$u \leftarrow u + \text{MG1}(f - Au; 0; \nu_1, \cdots, \nu_J).$$

end while

A slightly more general multigrid method.

1. Initialization of inputs

$$g_1 \leftarrow g, \quad u_1 \leftarrow \text{random}.$$

2. Smoothing and restriction
   - For $\ell = 1 : J$

– For $i = 1 : \nu_\ell$

$$u_\ell \leftarrow u_\ell + S_\ell * (g_\ell - A_\ell * u_\ell). \tag{8.130}$$

– Form restricted residual and set initial guess:

$$u_{\ell+1,0} \leftarrow \Pi_\ell^{\ell+1} u_\ell, \quad g_{\ell+1} \leftarrow R_\ell *_2 (g_\ell - A_\ell * u_\ell) + A_{\ell+1} * u_{\ell+1}^0,$$

3. Prolongation with post-smoothing
   - For $\ell = J - 1 : 1$

$$u_\ell \leftarrow u_\ell + R_\ell *_2^\top (u_{\ell+1} - u_{\ell+1}^0).$$

   – For $i = 1 : \nu'_\ell$

$$u_\ell \leftarrow u_\ell + S'_\ell * (g_\ell - A_\ell * u_\ell)$$

4. Output

$$u_1$$

Theorem 22. *If A^ℓ, $R_\ell^{\ell+1}$ and $B^{\ell,i} = S^\ell$ are all linear operations as described in multigrid method in §8.3 and all $\sigma = id$ in Algorithm 35. Then Algorithm 24 is equivalent to Algorithm 35 with any choice of $\Pi_\ell^{\ell+1}$.*

Proof. Here we replace $u^{\ell,i}$ and f^ℓ by $\tilde{u}^{\ell,i}$ and $\tilde{f}^\ell$ in MgNet. What we want to prove are

$$\tilde{f}^\ell = f^\ell + A_\ell \tilde{u}^{\ell,0} \quad \text{and} \quad u^{\ell,i} = \tilde{u}^{\ell,i} - \tilde{u}^{\ell,0}, \tag{8.131}$$

with $u^{\ell,i}$, f^ℓ in Algorithm 24 and $\tilde{u}^{\ell,i}$, $\tilde{f}^\ell$ in Algorithm 35 for any choice of $\Pi_\ell^{\ell+1}$. We prove this result by induction.

- It is easy to check that $\ell = 1$ is right by taking $\theta = \mathrm{id}$.
- Once the above equation (8.131) is right for ℓ, let us prove the corresponded result for $\ell + 1$.
  – For $\tilde{f}^{\ell+1}$, as the definition in Algorithm 35, we have

$$\begin{aligned}
\tilde{f}^{\ell+1} &= R_\ell^{\ell+1}(\tilde{f}^\ell - A^\ell \tilde{u}^{\ell,\nu_\ell}) + A^{\ell+1}\tilde{u}^{\ell+1,0}, \\
&= R_\ell^{\ell+1}(f^\ell + A^\ell \tilde{u}^{\ell,0} - A^\ell \tilde{u}^{\ell,\nu_\ell}) + A^{\ell+1}\tilde{u}^{\ell+1,0} \\
&= R_\ell^{\ell+1}(f^\ell - A^\ell(\tilde{u}^{\ell,\nu_\ell} - u^{\ell,0})) + A^{\ell+1}\tilde{u}^{\ell+1,0} \\
&= R_\ell^{\ell+1}(f^\ell - A^\ell u^{\ell,\nu_\ell}) + A^{\ell+1}\tilde{u}^{\ell+1,0}, \\
&= f^{\ell+1} + A^{\ell+1}\tilde{u}^{\ell+1,0}.
\end{aligned}$$

  – For $u^{\ell+1,i}$, first we have

$$u^{\ell+1,0} = 0 = \tilde{u}^{\ell+1,0} - \tilde{u}^{\ell+1,0},$$

  then we prove

$$u^{\ell+1,i} = \tilde{u}^{\ell+1,i} - \tilde{u}^{\ell+1,0} \tag{8.132}$$

by induction for i.
We assume (8.132) holds for $0, 1, \cdots, i-1$. Let us miner $\tilde{u}^{\ell+1,0}$ in both sides of the smoothing process (10.3) in Algorithm 35. Then we have

$$\begin{aligned}
\tilde{u}^{\ell+1,i} - \tilde{u}^{\ell+1,0} &= \tilde{u}^{\ell+1,i-1} - \tilde{u}^{\ell+1,0} + B^{\ell+1,i}(\tilde{f}^{\ell+1} - A^{\ell+1}\tilde{u}^{\ell+1,i-1}),\\
&= \tilde{u}^{\ell+1,i-1} - \tilde{u}^{\ell+1,0} + B^{\ell+1,i}(f^{\ell+1} + A^{\ell+1}\tilde{u}^{\ell+1,0} - A^{\ell+1}\tilde{u}^{\ell+1,i-1}),\\
&= u^{\ell+1,i-1} + B^{\ell+1,i}(f^{\ell+1} - A^{\ell+1}u^{\ell+1,i-1}).
\end{aligned}$$

This is exact the smoothing process in Algorithm 24 as we take $B^{\ell+1,i} = S^{\ell+1}$.

□

8.9 ReLU multigrid method for nonnegative solution

Considering $f = (1, 1, ..., 1)^\top$,

Algorithm 29 $(u^1, u^2, \cdots, u^J) = \text{MG0}(f; u^0; J, \nu_1, \cdots, \nu_J)$

Set up

$$f^1 = f, \quad u^1 = u^0.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
  for $i = 1 : \nu_\ell$ **do**

$$u^\ell \leftarrow u^\ell + S^\ell * \text{Relu}((f^\ell - A_\ell * u^\ell)). \tag{8.133}$$

  end for
  Form restricted residual and set initial guess:

$$u^{\ell+1,0} \leftarrow 0, \quad f^{\ell+1} \leftarrow R *_2 (f^\ell - A_\ell * u^\ell), A_{\ell+1} = R *_2 A_\ell * (R*_2^\top).$$

end for

Algorithm 5 is not convergent.

Algorithm 30 $(u^1, u^2, \cdots, u^J) = \text{MG0}(f; u^0; J, \nu_1, \cdots, \nu_J)$

Set up

$$f^1 = f, \quad u^1 = u^0.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
  for $i = 1 : \nu_\ell$ **do**

$$u^\ell \leftarrow u^\ell + \text{Relu}(S^\ell * (f^\ell - A_\ell * u^\ell)). \tag{8.134}$$

  end for
  Form restricted residual and set initial guess:

$$u^{\ell+1,0} \leftarrow 0, \quad f^{\ell+1} \leftarrow R *_2 (f^\ell - A_\ell * u^\ell), A_{\ell+1} = R *_2 A_\ell * (R *_2^\top).$$

end for

Algorithm 6 is convergent, the iterative steps is list below. $l = 2$

J	MG	Algorithm 6
2	15	27
3	16	35
4	17	38

8.9.1 Π is interpolation

Not Convergent

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + \text{Relu} \circ B_{\ell,i}(f^\ell - A^\ell(u^{\ell,i-1})). \tag{8.135}$$

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + B_{\ell,i} \circ \text{Relu}(f^\ell - A^\ell(u^{\ell,i-1})). \tag{8.136}$$

Almost the same as without Relu

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + B_{\ell,i}(f^\ell - \text{Relu} \circ A^\ell(u^{\ell,i-1})). \tag{8.137}$$

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + B_{\ell,i}(f^\ell - A^\ell \circ \text{Relu}(u^{\ell,i-1})). \tag{8.138}$$

8.10 Multigrid methods for nonlinear problem

In classification, the key problem can be reduced to find the representation(feature) for high dimension image for classifying. Here we propose to solve the next unbalanced nonlinear system

$$A(u) = f, \tag{8.139}$$

for finding the suitable feature representation $u \in \mathbb{R}^c$ for image $f \in \mathbb{R}^n$. The rationality of system can be traced back to the low-dimension assumption that natural image need to be concentrated on a low-dimension manifold with respect to the pixel space.

Model problem

$$\begin{cases} -\nabla \cdot (a(u)\nabla u) = f, & x \in \Omega \\ u = 0 & \text{on} \partial\Omega \end{cases} \tag{8.140}$$

Define

$$(A(u), v) = ((a(u)\nabla u, \nabla v)). \tag{8.141}$$

Then we have

$$A(u) = f. \tag{8.142}$$

Now the discretization problem reads: Find $u_h \in V_h$ such that

$$(A_h(u_h), v_h) = ((a(u_h)\nabla u_h, \nabla v_h)) \quad \forall v_h \in V_h. \tag{8.143}$$

Recall multigrid method for problems

$$A_h u_h = f_h, \tag{8.144}$$

1. Fine grid smoothing: applying m times gradient descent iterations to obtain u^m.
2. Coarse grid correction: solving the residual equation restricted on the coarse grid $\mathcal{T}_{2h}$ to obtain
$$A_{2h}\epsilon_{2h} = Q_{2h}\gamma^h$$
with $\gamma^h = f_h - A_h u_h^m$.
3. Update: $u_h \leftarrow u_h^m + I_{2h}^h \epsilon_{2h}$.

Coarse grid correction for the linear case can be rewritten as:

$$A_{2h}\epsilon^{2h} = Q_{2h}(f_h - A_h u_h^m)$$

We write the above equation as

$$Q_{2h}(A_h(u_h^m + \epsilon_{2h}) - A_h(u_h^m) = Q_{2h}(f_h - A_h u_h^m)$$

Next we think about the nonlinear multigrid methods. **Question:** Given $u_h^0 \in V_h$, $u_h^0 \approx u_h$, we need to find a correction $e_{2h} \in V_{2h}$, s.t.

1. $u_h^1 = u_h^0 + e_{2h} \approx u_h$;
2. $e_{2h} = 0$ if $u_h^0 = u_h$;
3. The solution of e_{2h} is obtained by solving the coarse grid equation

$$A_{2h}(w_{2h}) = g_{2h}.$$

Solution: $Q_{2h}A_h(u_h^0 + e_{2h}) = f_{2h}$.
(i) $Q_{2h}[A_h(u_h^0 + e_{2h}) - A_h(u_h^0)] = Q_{2h}[f_{2h} - A_h(u_h^0)]$.
Now we use the approximation

$$Q_{2h}A_h(u_h^0 + e_{2h}) \approx A_{2h}(u_{2h}^0 + e_{2h}). \tag{8.145}$$

Since we need to find approximation of $Q_{2h}A_h(u_h^0)$, s.t. $e_{2h} = 0$ if $h_h^0 = u_h$;
Then we must have

$$Q_{2h}A_h(u_h^0) \approx A_{2h}(u_{2h}^0).$$

In summary, we can have twogrid method for nonlinear problem $A_h(u_h) = f_h$ as follows

1. Fine grid smoothing:
$$u_h \leftarrow u_h + S_h(f_h - A_(u_h))$$
such as Newton method.
2. Coarse grid correction:
$$u_{2h,0} = \Pi_\ell^{\ell+1} u_h, \quad A_{2h}(u_{2h}) = Q_{2h}(f_h - A_h(u_h) + A_{2h}(u_{2h,0}).$$
3. Update: $u_h \leftarrow u_h + u_{2h} - u_{2h,0}$.

To solve the overdetermined nonlinear system, we can try the multilevel ideas with smoothing is fine level, and truncated it into coarse level by recursion. One strategy to involve the multi-scale idea is to "soothing" in the fine level, and restrict it as a good approximation in the coarse level, this idea can be found in many literatures especially for multigrid methods in optimization [21, 15]. So, there is a more general nonlinear multigrid scheme - fully approximation scheme (FAS) [1, 22], which can be considered as the generalization of linear multigrid scheme 24. Here we show a FAS scheme with Slash cycle as

Algorithm 31 $u =$ Bslash-FAS$(u^{1,0}, f, J, m_1, \cdots, m_J)$

Initialization

$$f^1 = f.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
Nonlinear relaxation on level ℓ:
for $i = 1 : m_j$ **do**

$$u^{\ell,i} = u^{\ell,i-1} + [\nabla F^\ell(u^{\ell,i-1})]^{-1}(f^\ell - F^\ell(u^{\ell,i-1})).$$

end for

Form the initial guess and right side term for level $\ell+1$:

$$u^{\ell+1,0} = \Pi_\ell^{\ell+1} u^{\ell,m_\ell}, \quad f^{\ell+1} = R_\ell^{\ell+1}(f^\ell - F^\ell(u^{\ell,m_\ell})) + F^{\ell+1}(u^{\ell+1,0}).$$

end for
Prolongation and correction from coarse to fine level
for $\ell = J-1:1$ **do**
Form error in coarse level

$$e^{\ell+1} = u^{\ell+1,m_{\ell+1}} - u^{\ell+1,0}.$$

Correction by using error in coarse level

$$u^{\ell,m_\ell} \leftarrow u^{\ell,m_\ell} + P_{\ell+1}^\ell e^{\ell+1}.$$

end for

If the problem in (8.139) is linear, then we have the next theorem to show that this FAS scheme is consist with the classical multigrid methods for linear systems.

Theorem 23. *If $F(u)$ in (8.139) is a linear operation. Then Algorithm 31 is equivalent to Algorithm 22 with any choice of $\Pi_\ell^{\ell+1}$.*

9

Convolutional Neural Networks

9.1 Convolutional operations

9.1.1 Images as matrix

An image can be viewed as a piecewise constant function on a grid. Images with different resolutions can then be viewed as functions on grids of different sizes. The use of such multiple-grids is a main technique used in the standard multigrid method for solving discretized partial differential equations, and it can also be interpreted as a main ingredient used in convolutional neural networks (CNN) for image calssification.

An image can be viewed as a function on a grid [11] on a rectangle domain $\Omega \in \mathcal{R}^2$. Without loss of generality, we assume that the grid, $\mathcal{T}$, is of size

$$m = 2^s, \quad n = 2^t$$

for some integers $s, t \geq 1$. Starting from $\mathcal{T}_1 = \mathcal{T}$, we consider a sequence of coarse grids with $J = \min(s, t)$ (as depicted in Fig. 9.1.1 with $J = 4$):

$$\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_J \tag{9.1}$$

such that $\mathcal{T}_\ell$ consist of $m_\ell \times n_\ell$ grid points, with

$$m_\ell = 2^{s-\ell+1}, \quad n_\ell = 2^{t-\ell+1}. \tag{9.2}$$

Here, please note that each element in this grid can be viewed as a pixel or an image or an element in a matrix.

9.1.2 Convolution operation with one channel

For simplicity of exposition, we denote

$$m = m_1 = 2^s, \quad n = n_1 = 2^t. \tag{9.3}$$

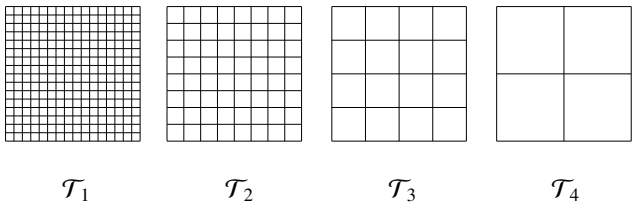


Fig. 9.1. multilevel grids for piecewise constant functions (images)

Definition 12. *A convolution defined on $\mathbb{R}^{m\times n}$ is a linear mapping $K* : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{m\times n}$ defined with padding, for any $g \in \mathbb{R}^{m\times n}$ by:*

$$[K * g]_{i,j} = \sum_{p,q=-k}^{k} K_{p,q} g_{i+p,j+q}, \quad i = 1 : m, j = 1 : n. \tag{9.4}$$

Here we note that the indices for the entries in K are given un a special way. For example, if $k = 1, K \in \mathbb{R}^{3\times 3}$, and

$$K = \begin{pmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,1} \\ K_{0,-1} & K_{0,0} & K_{0,1} \\ K_{1,-1} & K_{1,0} & K_{1,1} \end{pmatrix},$$

for we may have the following 2D Laplacian kernel

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}. \tag{9.5}$$

The coefficients in (9.17) constitute a kernel matrix

$$K \in \mathbb{R}^{(2k+1)\times(2k+1)}, \tag{9.6}$$

where k is often taken as a small integer. Here padding means how $g_{i+p,j+q}$ is defined when $(i + p, j + q)$ is out of $1 : m$ or $1 : n$. The following three choices are often used

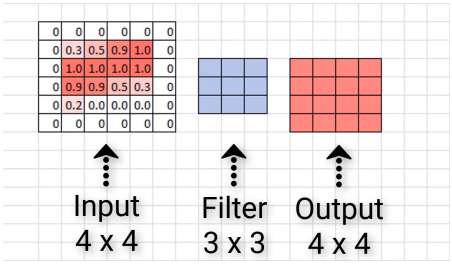
$$g_{i+p,j+q} = \begin{cases} 0, & \text{zero padding,} \\ f_{(i+p) \pmod m,(s+q) \pmod n}, & \text{periodic padding,} \\ f_{|i-1+p|,|j-1+q|}, & \text{reflected padding,} \end{cases} \tag{9.7}$$

if

$$i + p \notin \{1, 2, \ldots, m\} \text{ or } j + q \notin \{1, 2, \ldots, n\}. \tag{9.8}$$

Here $d \pmod m \in \{1, \cdots, m\}$ means the remainder when d is divided by m.

Here is a diagram for convolution with one channel (and also stride one).



9.1.3 Convolution with stride (one channel)

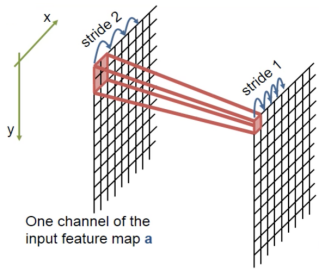
Definition 13. *Convolution with stride* 2 *is defined as*

$$[K *_2 g]_{i,j} = \sum_{p,q=-k}^{k} K_{p,q} g_{2i+p-1,2j+q-1}, \quad i = 1 : \lfloor \frac{m+1}{2} \rfloor, j = 1 : \lfloor \frac{n+1}{2} \rfloor. \tag{9.9}$$

We note that, in general, for any given integer $s \geq 1$, a convolution with stride s for $g \in \mathbb{R}^{m \times n}$ can be defined as:

$$[K *_s g]_{i,j} = \sum_{p,q=-k}^{k} K_{p,q} g_{s(i-11)+p+1,s(j-1)+q+1}, \quad i = 1 : \lfloor \frac{m+1}{s} \rfloor, j = 1 : \lfloor \frac{n+1}{s} \rfloor. \tag{9.10}$$

Here $\lfloor \frac{m}{s} \rfloor$ denotes the biggest integer that less than $\frac{m}{s}$. The following is a diagram for stride 2.



Lemma 28. *The convolution with stride* 2 *can be written as:*

$$K *_2 g = \mathcal{S}(K * g), \tag{9.11}$$

where $\mathcal{S}$ *is a stride operator defined by:*

$$\mathcal{S} : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{\frac{m+1}{2}\times\frac{n+1}{2}}, \tag{9.12}$$

with

$$[\mathcal{S}(g)]_{i,j} = g_{2i-1,2j-1}, \quad i = 1 : \lfloor\frac{m+1}{2}\rfloor, j = 1 : \lfloor\frac{n+1}{2}\rfloor. \tag{9.13}$$

Example 7. The so-called average pooling with kernel size 3×3 and stride 2 means

$$K*_2, \tag{9.14}$$

where

$$K = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \tag{9.15}$$

9.1.4 Convolutional operations with multi-channel

One important class of linear mapping is the so-called convolution:

$$\theta : \mathbb{R}^{c\times m\times n} \mapsto \mathbb{R}^{h\times m\times n},$$

where $m \times n$ is called the spatial dimension or resolution, c and h are corresponding to input and output channels. The operation is defined by

$$[\theta(f)]_s = \sum_{t=1}^{c} K_{s,t} * [f]_t + b_s\mathbf{1} \in \mathbb{R}^{m\times n}, \quad s = 1 : h, \tag{9.16}$$

where $\mathbf{1} \in \mathbb{R}^{m\times n}$ is a $m \times n$ matrix with all elements being 1, and for $[f]_t \in \mathbb{R}^{m\times n}$ represent for the t-th channel

$$[K_{s,t} * [f]_t]_{i,j} = \sum_{p,q=-k}^{k} K_{s,t;p,q} f_{t;i+p,j+q}, \quad i = 1 : m, j = 1 : n. \tag{9.17}$$

The coefficients kernel $K_{s,t}$ in (9.17) constitute a kernel matrix

$$K_{s,t} \in \mathbb{R}^{(2k+1)\times(2k+1)}, \tag{9.18}$$

where k is often taken as small integers.

Here a more compact notation for multi-channel convolution can be written as

$$\theta(f) = K * f + \mathbf{b} \tag{9.19}$$

where

$$f = \begin{pmatrix} [f]_1 \\ [f]_2 \\ \vdots \\ [f]_c \end{pmatrix}, \quad K = \begin{pmatrix} K_{1,1} & K_{1,2} & \cdots & K_{1,c} \\ K_{2,1} & K_{2,2} & \cdots & K_{2,c} \\ \vdots & \vdots & \ddots & \vdots \\ K_{h,1} & K_{h,2} & \cdots & K_{h,c} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1\mathbf{1} \\ b_2\mathbf{1} \\ \vdots \\ b_h\mathbf{1} \end{pmatrix} = b \otimes \mathbf{1}. \tag{9.20}$$

Furthermore, we have the following natural extension of convolution with stride for multi-channel by

$$[\theta(f)]_s = \sum_{t=1}^{c} K_{s,t} *_2 [f]_t + b_s\mathbf{1} \in \mathbb{R}^{\tilde{m}\times\tilde{n}}, \quad s = 1 : h, \tag{9.21}$$

where

$$\tilde{m} = \lfloor \frac{m+1}{2} \rfloor, \quad \tilde{n} = \lfloor \frac{n+1}{2} \rfloor, \tag{9.22}$$

9.1.5 Pooling operation in CNNs

Finally, we introduce another type of important operation in CNNs – pooling. The key purpose for pooling operator is to reduce the spatial resolution of images (features) in a typical CNN models. Basically, pooling is an operator

$$T : \mathbb{R}^{c_1\times m_1\times n_1} \mapsto \mathbb{R}^{c_2\times m_2\times n_2}. \tag{9.23}$$

where

$$m_2 = \lfloor \frac{m+1}{s} \rfloor, \quad n_2 = \lfloor \frac{n+1}{s} \rfloor, \tag{9.24}$$

for any choice of $c_2 \geq 1$. Here s is also called the stride in pooling operations. There are generally two types of pooling

Convolution with stride s as pooling

In this case, it often happens that

$$T = R*_s, \quad (s = 2 \text{ for the main case }). \tag{9.25}$$

Here R can be learned or fixed such as average pooling as we discussed before.

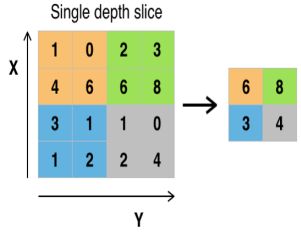
Nonlinear pooling

The most commonly used nonlinear pooling is called max-pooling, a max pooling with kernel size $(2k+1)\times(2k+1)$ and stride s is is defined as

$$[R_{\max}(f)]_{t;i,j} = \max\{f_{t;si+p-1,sj+q-1} \mid -k \le p,q \le k\}, \tag{9.26}$$

here t means channel and $c_2 = c_1$ in this case.

Here is an example for max-pooling with kernel size 2×2 and stride 2.



9.2 Examples of convolution filters and performance

In this section, we will give a brief description how convolution operations are used for image processing. One useful description can be found in the following link:

http://aishack.in/tutorials/image-convolution-examples/

Convolutions is a technique for general signal processing. People studying electrical/electronics will tell you the near infinite sleepless nights these convolutions have given them. Entire books have been written on this topic. And the questions and theorems that need to be proved are [insurmountable]. But for computer vision, we'll just deal with some simple things.

A convolution lets you do many things, like calculate derivatives, detect edges, apply blurs, etc. A very wide variety of things. And all of this is done with a "convolution kernel".

9.2.1 Calculation with convolutions

The most direct way to compute a convolution would be to use multiple for loops. But that causes a lot of repeated calculations. And as the size of the image and kernel increases, the time to compute the convolution increases too (quite drastically).

Techniques haves been developed to calculate convolutions rapidly. One such technique is using the Discrete Fourier Transform. It converts the entire convolution operation into a simple multiplication. Fortunately, you don't need to know the math to do this in OpenCV. It automatically decides whether to do it in frequency domain (after the DFT) or not.

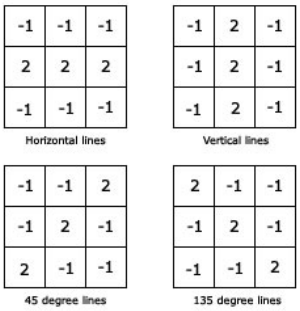
9.2.2 Image convolution examples

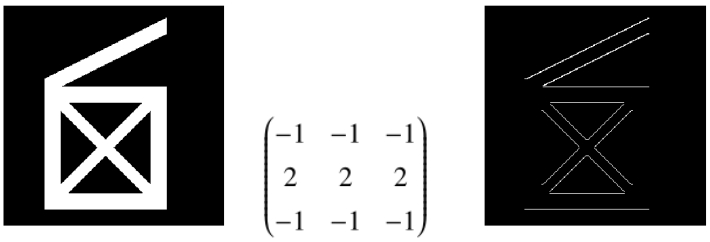
A convolution is very useful for signal processing in general. There is a lot of complex mathematical theory available for convolutions. For digital image processing, you don't have to understand all of that. You can use a simple matrix as an image convolution kernel and do some interesting things!

9.2.3 Line detection by 1D Laplacian

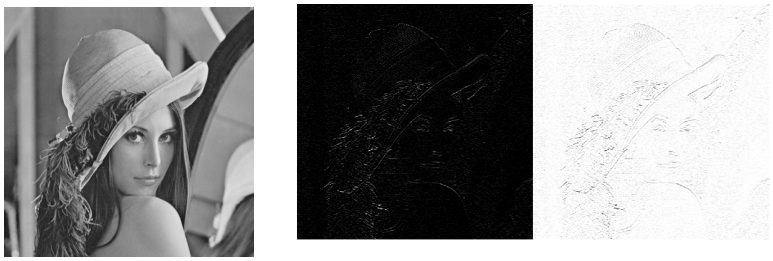
With image convolutions, you can easily detect lines. Here are four convolutions to detect horizontal, vertical and lines at 45 degrees:

Here's 0,90,45,135 lines detection that I got on an image:





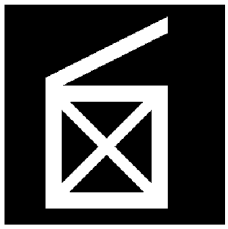
(a) image (b) filter (c) result



(d) image (e) result

Fig. 9.2. A horizontal line detection done with convolutions

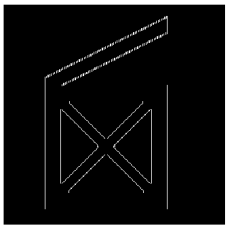
In Lena,the black background is the original result, the white background is obtained by subtracting the original result from 255, the same below.



(a) image

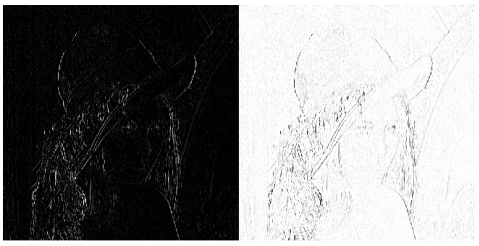
$$\begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$$

(b) filter



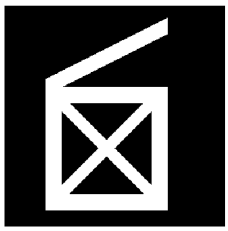
(c) result

(d) image



(e) result

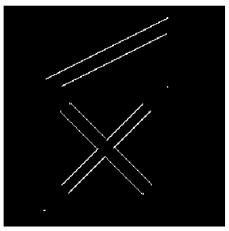
Fig. 9.3. A vertical line detection done with convolutions



(a) image

$$\begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix}$$

(b) filter

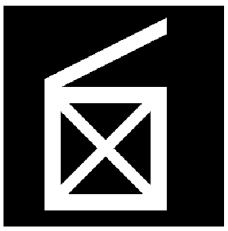


(c) result

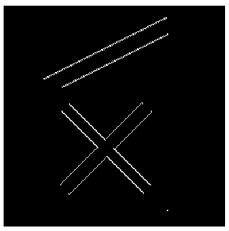


(d) image (e) result

Fig. 9.4. A 45 degress line detection done with convolutions



$$\begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$



(a) image (b) filter (c) result



(d) image (e) result

Fig. 9.5. A 135 degress line detection done with convolutions

9.2.4 Edge detection by 2D Laplacian operator

The laplacian is the second derivative of the image. It is extremely sensitive to noise, so it isn't used as much as other operators. Unless, of course you have specific requirements.

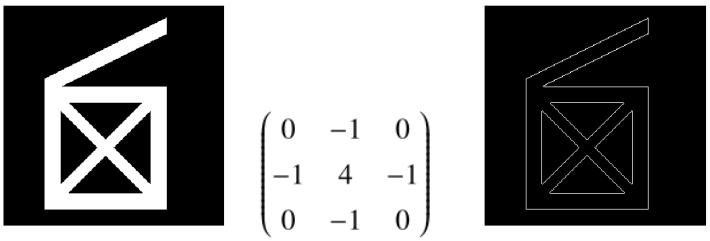
0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

Here's the result with the convolution kernel without diagonals:



$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

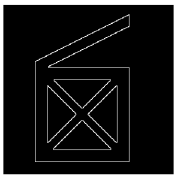
Fig. 9.6. A laplace operator done with convolutions

The result with the convolution kernel with diagonals:

(a) image

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

(b) filter



(c) result

(d) image

(e) result

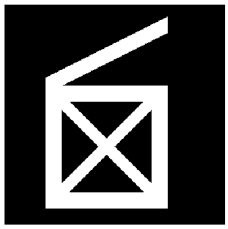
Fig. 9.7. A laplace operator include diagonals done with convolutions

9.2.5 The Laplacian of Gaussian

The laplacian alone has the disadvantage of being extremely sensitive to noise. So, smoothing the image before a laplacian improves the results we get. This is done with a 5x5 image convolution kernel.

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

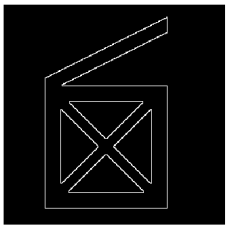
The result on applying this image convolution was:



(a) image

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

(b) filter



(c) result

(d) image

(e) result

Fig. 9.8. A Laplacian of Gaussian operator done with convolutions

9.2.6 Other examples with ReLU activation

(a) input image

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

(b) filter

(c) convolution result

(d) result after ReLU

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(e) filter

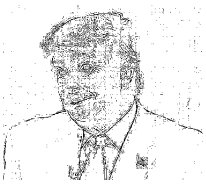
(f) result after average

(g) image

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

(h) filter

(i) filter



(j) ReLU

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(k) filter

(l) average

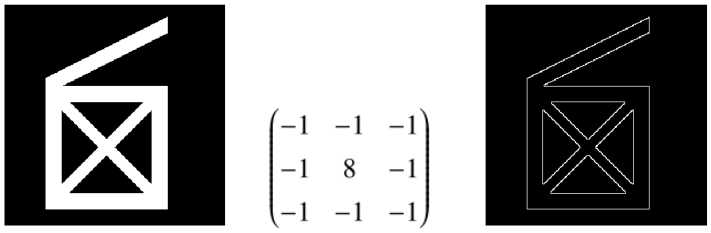
9.2.7 Some other examples

Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to ”combine” the results is to merge the convolution kernels. The new image convolution kernel looks like this:

-1	-1	-1
-1	8	-1
-1	-1	-1

Below result I got with edge detection:



$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

(m) image (n) filter (o) result

(p) image (q) result

Fig. 9.9. A edge detection done with convolutions

The Sobel Edge Operator

The above operators are very prone to noise. The Sobel edge operators have a smoothing effect, so they're less affected to noise. Again, there's a horizontal component and a vertical component.

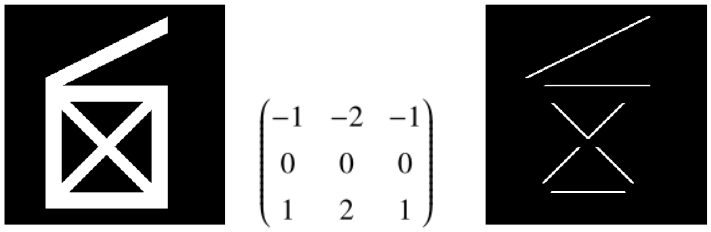
-1	-2	-1
0	0	0
1	2	1

Horizontal

-1	0	1
-2	0	2
-1	0	1

Vertical

On applying horizontal component in image , the result was:



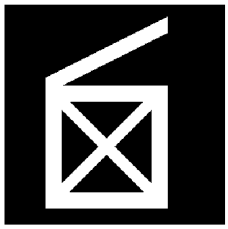
$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

(a) image (b) filter (c) result

(d) image (e) result

Fig. 9.10. A horizontal sobel edge operator done with convolutions

On applying vertical component in image , the result was:



(a) image

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

(b) filter

(c) result

(d) image

(e) result

Fig. 9.11. A vertical sobel edge operator done with convolutions

Simple box blur

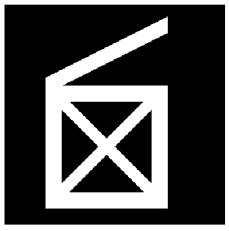
[1] Here's a first and simplest. This convolution kernel has an averaging effect. So you end up with a slight blur. The image convolution kernel is:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Note that the sum of all elements of this matrix is 1.0. This is important. If the sum is not exactly one, the resultant image will be brighter or darker.

Here's a blur that I got on an image:

[1] The following examples are from the website, http://aishack.in/tutorials/image-convolution-examples/



(a) image

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(b) filter

(c) result

(d) image

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

(e) filter

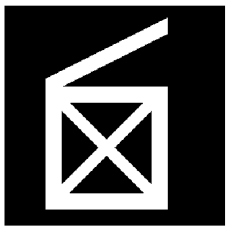
(f) result

Fig. 9.12. A simple blur done with convolutions

Gaussian blur

Gaussian blur has certain mathematical properties that makes it important for computer vision. And you can approximate it with an image convolution. The image convolution kernel for a Gaussian blur is:

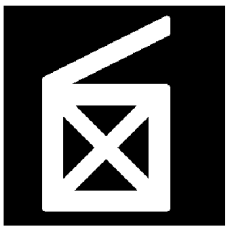
Here's a result that I got:



(a) image

$$\begin{pmatrix} 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 5 & 32 & 100 & 100 & 100 & 32 & 5 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix}$$

(b) filter



(c) result

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

(d) image

$$\begin{pmatrix} 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 5 & 32 & 100 & 100 & 100 & 32 & 5 \\ 0 & 18 & 64 & 100 & 64 & 18 & 0 \\ 0 & 5 & 18 & 32 & 18 & 5 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix}$$

(e) filter

(f) result

Fig. 9.13. A Gaussian blur done with convolutions

9.2.8 Summary

You got to know about some important operations that can be approximated using an image convolution. You learned the exact convolution kernels used and also saw an example of how each operator modifies an image. I hope this helped!

9.3 Some classic CNN models

In this section, we will use these convolutional operations introduced above to give a brief description of some classic convolutional neural network (CNN) models. Firstly, CNNs are actually a class of special DNN models. Let us recall the DNN structure as:

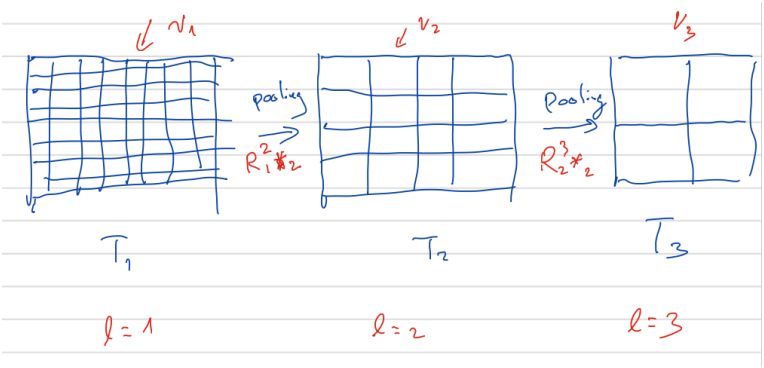
$$\begin{cases} f^0(x) &= x \\ f^\ell(x) &= \sigma(\theta^\ell(f^{\ell-1})) \quad \ell = 1:L\,, \\ f(x) &= W^L f^L + b^L \end{cases} \tag{9.27}$$

where

$$(\theta^\ell(f^{\ell-1}) = W^\ell f^{\ell-1}(x) + b^\ell. \tag{9.28}$$

So the key features of CNNs is

1. Replace the general linear mapping to be convolution operations with multi-channel.
2. Use multi-resolution of images as shown in the next diagram.



Then we will introduce some classical architectures in convolution neural networks.

9.3.1 LeNet-5, AlexNet and VGG

The LeNet-5 [12], AlexNet [11] and VGG [18] can be written as:

Algorithm 32 h = Classic CNN$(f; J, \nu_1, \cdots, \nu_J)$

1: Initialization: $f^{1,0} = f_{\text{in}}(f)$.
2: **for** $\ell = 1 : J$ **do**
3: **for** $i = 1 : \nu_\ell$ **do**

4: Basic Block:

$$f^{\ell,i} = \sigma\left(\theta^{\ell,i} * f^{\ell,i-1}\right) \tag{9.29}$$

5: **end for**
6: Pooling(Restriction):

$$f^{\ell+1,0} = R_\ell^{\ell+1} *_2 f^{\ell,\nu_\ell} \tag{9.30}$$

7: **end for**
8: Final average pooling layer: $h = R_{\text{ave}}(f^{L,\nu_\ell})$.

Here $R_\ell^{\ell+1}*_2$ represents for the pooling operation to sub-sampling these tensors into coarse spatial level (lower resolution). Here we use $R_\ell^{\ell+1}*_2$ to stand for the pooling operation. In general we can also have

- average pooling: fixed kernels such as

$$R_\ell^{\ell+1} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{9.31}$$

- Max pooling $R_{\max}$ as discussed before.

In these three classic CNN models, they still need some extra fully connected layers after h as the output of CNNs. After few layers of fully connected layers, the model is completed by following a multi-class logistic regression model.

These fully connected layers are removed in ResNet to be described below.

9.3.2 ResNet

The original ResNet developed in [7] is one of the most popular CNN architectures in image classification problems.

Algorithm 33 $h = \text{ResNet}(f; J, \nu_1, \cdots, \nu_J)$

1: Initialization: $r^{1,0} = f_{\text{in}}(f)$.
2: **for** $\ell = 1 : J$ **do**
3: **for** $i = 1 : \nu_\ell$ **do**
4: Basic Block:

$$r^{\ell,i} = \sigma\left(r^{\ell,i-1} + A^{\ell,i} * \sigma \circ B^{\ell,i} * r^{\ell,i-1}\right). \tag{9.32}$$

5: **end for**
6: Pooling(Restriction):

$$r^{\ell+1,0} = \sigma\left(R_\ell^{\ell+1} *_2 r^{\ell,\nu_\ell} + A^{\ell+1,0} \circ \sigma \circ B^{\ell+1,0} *_2 r^{\ell,\nu_\ell}\right). \tag{9.33}$$

7: **end for**

8: Final average pooling layer: $h = R_{\text{ave}}(r^{L,\nu_\ell})$.

Here $f_{\text{in}}(\cdot)$ may depend on different data set and problems such as $f_{\text{in}}(f) = \sigma \circ \theta^0 * f$ for CIFAR [10] and $f_{\text{in}}(f) = R_{\max} \circ \sigma \circ \theta^0 * f$ for ImageNet [3] as in [8]. In addition $r^{\ell,i} = r^{\ell,i-1} + A^{\ell,i} * \sigma \circ B^{\ell,i} * \sigma(r^{i-1})$ is often called the basic ResNet block. Here, $A^{\ell,i}$ with $i \geq 0$ and $B^{\ell,i}$ with $i \geq 1$ are general 3×3 convolutions with zero padding and stride 1. In pooling block, $*_2$ means convolution with stride 2 and $B^{\ell,0}$ is taken as the 3×3 kernel with same output channel dimension of $R_\ell^{\ell+1}$ which is taken as 1×1 kernel and called as projection operator in [8]. During two consecutive pooling blocks, index ℓ means the fixed resolution or we ℓ-th grid level as in multigrid methods. Finally, R_{ave} ($R_{\max}$) means average (max) pooling with different strides which is also dependent on datasets and problems.

9.3.3 pre-act ResNet

The pre-act ResNet [8] shares a similar structure with ResNet.

Algorithm 34 h = pre-act ResNet$(f; J, \nu_1, \cdots, \nu_J)$

1: Initialization: $r^{1,0} = f_{\text{in}}(f)$.
2: **for** $\ell = 1 : J$ **do**
3: **for** $i = 1 : \nu_\ell$ **do**
4: Basic Block:

$$r^{\ell,i} = r^{\ell,i-1} + A^{\ell,i} * \sigma \circ B^{\ell,i} * \sigma(r^{\ell,i-1}). \tag{9.34}$$

5: **end for**
6: Pooling(Restriction):

$$r^{\ell+1,0} = R_\ell^{\ell+1} *_2 r^{\ell,\nu_\ell} + A^{\ell+1,0} \circ \sigma \circ B^{\ell+1,0} *_2 \sigma(r^{\ell,\nu_\ell}). \tag{9.35}$$

7: **end for**
8: Final average pooling layer: $h = R_{\text{ave}}(r^{L,\nu_\ell})$.

Here pre-act ResNet share almost the same setup with ResNet.

The only difference between ResNet and pre-act ResNet can be viewed as putting a σ in different places. The connection of those three models are often shown with next diagrams:

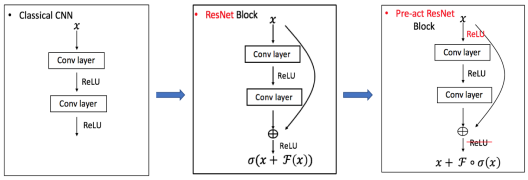


Fig. 9.14. Comparison of CNN Structures

Without loss of generality, we extract the key feedforward steps on the same grid in different CNN models as follows.

Classic CNN

$$f^{\ell,i} = \xi^i \circ \sigma(f^{\ell,i-1}) \quad \text{or} \quad f^{\ell,i} = \sigma \circ \xi^i(f^{\ell,i-1}). \tag{9.36}$$

ResNet

$$r^{\ell,i} = \sigma(r^{\ell,i-1} + A^{\ell,i} \circ \sigma \circ B^{\ell,i}(r^{\ell,i-1})). \tag{9.37}$$

pre-act ResNet

$$r^{\ell,i} = r^{\ell,i-1} + A^{\ell,i} \circ \sigma \circ B^{\ell,i} \circ \sigma(r^{\ell,i-1}). \tag{9.38}$$

10

MgNet: a Unified Framework for CNN and MG

10.1 MgNet: a new network structure

In this section, we introduce a new neural network structure, named as MgNet, motivated by the multigrid algorithm, Algorithm 24 and its nonlinear version in Algorithm 31, as discussed in the previous section.

Here we recall the most important two structures in multigrid

1. iterative scheme (for linear system)

$$u^{\ell,i} = u^{\ell,i-1} + B^{\ell,i}(f^\ell - A^\ell * u^{\ell,i-1}). \tag{10.1}$$

2. interpolation and restriction

$$u^{\ell+1,0} = \Pi_\ell^{\ell+1} *_2 u^\ell, \quad f^{\ell+1} = R_\ell^{\ell+1} *_2 (f^\ell - A^\ell(u^\ell)) + A^{\ell+1} * u^{\ell+1,0}. \tag{10.2}$$

Considering the fine to coarse process of multigrid with the aforementioned two structures in (10.1) and (10.2). We are now in a position to state the main algorithm, namely MgNet by just put some nonlinear activation function σ in some places.

Algorithm 35 $u^J = \mathrm{MgNet}(f; J, \nu_1, \cdots, \nu_J)$

Initialization: $f^1 = \theta(f)$, $u^{1,0} = 0$
for $\ell = 1 : J$ **do**
  for $i = 1 : \nu_\ell$ **do**

$$u^{\ell,i} = u^{\ell,i-1} + \sigma \circ B^{\ell,i} * \sigma(f^\ell - A^\ell * u^{\ell,i-1}). \tag{10.3}$$

  end for
  Note $u^\ell = u^{\ell,\nu_\ell}$

$$u^{\ell+1,0} = \Pi_\ell^{\ell+1} *_2 u^\ell \tag{10.4}$$

$$f^{\ell+1} = R_\ell^{\ell+1} *_2 (f^\ell - A^\ell(u^\ell)) + A^{\ell+1} * u^{\ell+1,0}. \tag{10.5}$$

end for

The main steps in MgNet can be understood as solving the following data-feature mappings in each grid ℓ:

$$A^\ell * u^\ell = f^\ell, \quad \ell = 1 : J, \tag{10.6}$$

where

$$f^\ell \in \mathbb{R}^{c_\ell \times m_\ell \times n_\ell}, \tag{10.7}$$

and

$$u^\ell \in \mathbb{R}^{h_\ell \times m_\ell \times n_\ell}, \tag{10.8}$$

with constrain

$$u > 0. \tag{10.9}$$

More details about this basic assumption in image classification can be found in [5].

Here, the next diagram gives a brief illustration for the above structure.

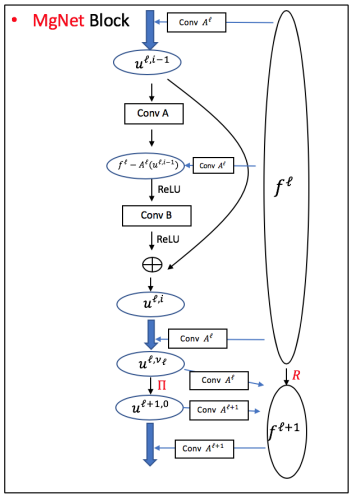


Fig. 10.1. Structure of MgNet

The first property of MgNet is that it recovers the multigrid methods.

Despite of the simplicity look of Algorithm 35, there are rich mathematical structures and variants which we briefly discuss below.

10.1.1 Initialization: feature space channels

Initially for $\ell = 1$, we take $m_1 = m$ and $n_1 = n$ and we may define the linear mapping

$$\theta : \mathbb{R}^{c\times m\times n} \mapsto \mathbb{R}^{c_1\times m_1\times n_1}, \tag{10.10}$$

to obtain $f^1 = \theta(f)$ with c given in (1.2) changed to the channel of the initial data space to c_1. Usually

$$c_1 \geq c. \tag{10.11}$$

One possibility is that we choose $c_1 = c$. In this case, we choose θ =identity. But in general, we may need to choose $c_1 \gg c$. One possible advantage of preprocessing the RGB ($c = 3$) to different color spaces is that we can better choose what kind of features the CNN can detect, and under what conditions those detections will be invariant.

One possibility of understanding and modifying this step is to decompose the data f into a number of more specialized data

$$f = \sum_{k=1}^{c_1} \xi_k f_k^1 = \xi^T f^1. \tag{10.12}$$

We may use some knowledge from image processing or physics to design a procedure to obtain the right decomposition of (10.12), or we can just train it. Conceivably, we may view $f^1 = \theta(f)$ as a special approximation solution of (10.12) with the same sparsity pattern to ξ.

10.1.2 Extracted Units: u^ℓ and channels

The first new feature and the main new ingredient in the proposed neural network is the introduction of feature variables u^ℓ in (10.8), which will be known as the extracted units.

We emphasize that the extracted-units u^ℓ and the data f^ℓ can have different numbers of channels:

$$u^\ell \in \mathbb{R}^{c_{u,\ell}\times m_\ell\times n_\ell}, \quad f^\ell \in \mathbb{R}^{c_{f,\ell}\times m_\ell\times n_\ell} \tag{10.13}$$

One possibility is that the number of channels for both u and f remain unchanged in different grids:

$$c_{f,\ell} = c_f, \quad \ell = 1 : J, \tag{10.14}$$

and

$$c_{u,\ell} = c_u, \quad \ell = 1 : J. \tag{10.15}$$

Both c_f and c_u are two super-parameters that need to be tuned, and we may even take $c_u = c_f$.

10.1.3 Poolings: $\Pi_{\ell+1}^{\ell}$ and $R_{\ell+1}^{\ell}$

The pooling $\Pi_{\ell+1}^{\ell}$ in (10.4) and $R_{\ell+1}^{\ell}$ in (10.5) are in general different. They can be trained in general, but they may be a priori chosen.

There are many different possibilities to choose $\Pi_{\ell+1}^{\ell}$. The simplest choice of $\Pi_{\ell+1}^{\ell}$ is

$$\Pi_{\ell+1}^{\ell} = 0. \tag{10.16}$$

A more sophisticated choice can be obtained by considering an interpolation from fine grid to coarse (that, for example preserves linear function locally). Namely

$$\Pi_{\ell+1}^{\ell} = \bar{\Pi}_{\ell+1}^{\ell} \otimes I_{c_\ell \times c_\ell} \tag{10.17}$$

with $\bar{\Pi}_{\ell+1}^{\ell}$ given in finite element methods.

10.1.4 Data-feature mapping: A^{ℓ}

The second new feature of MgNet is that this data-feature mapping only depends on the grid $\mathcal{T}_\ell$, and it does not depend on layers within the same grid. This amounts to a significant saving of the number of parameters. In comparison, the existing CNN, such as pre-act ResNet, can be interpreted as a network related to the case that A^{ℓ} is replaced by $A^{\ell,i}$, namely

$$u^{\ell,i} = u^{\ell,i-1} + \sigma \circ B^{\ell,i} * \sigma(f^{\ell} - A^{\ell,i} * u^{\ell,i-1}). \tag{10.18}$$

The underlying convolution kernels can be different on different grids and they can all be trained.

10.1.5 Feature extractors: $\sigma \circ B^{\ell,i} * \sigma$

Here we adopt the feature extractor as:

$$\sigma \circ B^{\ell,i} * \sigma. \tag{10.19}$$

Other than the level dependent extractors, the following different strategies can be used

Constant Extractors : $B^{\ell,i} = B^{\ell}$ for $i = 1 : \nu_\ell$
Scaled Extractors :$B^{\ell,i} = \alpha_i B^{\ell}$ for $i = 1 : \nu_\ell$
Variable Extractors : $B^{\ell,i}$

This brief framework gives us the basic principle on designing a CNN models for classification. All models are seen as the special choice of data-feature mapping A^{ℓ}, feature extractors $B^{\ell,i}$ and the pooling operators $\Pi_{\ell+1}^{\ell}$ with $R_{\ell+1}^{\ell}$.

10.2 Variants and generalizations of MgNet

The MgNet model algorithm is one very basic and it can be generalized in many different ways. It can also be used as a guidance to modify and extend many existing CNN models.

The following result show how MgNet is related to he pre-act ResNet [8].

Theorem 24. *The MgNet model Algorithm 35, admits the following identities*

$$r^{\ell,i} = f^{\ell,i-1} - A^{\ell} \circ \sigma \circ B^{\ell,i} \circ \sigma(r^{\ell,i-1}), \quad i = 1 : \nu_\ell, \tag{10.20}$$

where

$$r^{\ell,i} = f^{\ell} - A^{\ell} * u^{\ell,i}. \tag{10.21}$$

Furthermore, (10.20) *represents pre-act ResNet [8] as shown before.*

Proof. Because of the linearity of A^ℓ and invariant within the same grid ℓ, we can apply A^ℓ on both sides of (10.3) and minus with f^ℓ, thus we have

$$f^{\ell} - A^{\ell} * u^{\ell,i} = f^{\ell} - A^{\ell} * u^{\ell,i-1} - A^{\ell} * \sigma \circ B^{\ell,i} \circ \sigma(f^{\ell} - A^{\ell} * u^{\ell,i-1}).$$

This finish the proof with definition in (10.21). □

The above result is very simple but critically important. In view of Theorem 24, it shows how multigrid and CNN are intimately related. Furthermore, it provides a different version of iResNet, which can be viewed as the dual version of the original pre-act ResNet. This relation is quit similar with the dual relation of u and f in multigrid method [25].

Lemma 29. *The ResNet [7] step as in* (9.37) *admits the following relation:*

$$\tilde{r}^{\ell,i+1} = \sigma(\tilde{r}^{\ell,i}) - A^{\ell,i} * \sigma \circ B^{\ell,i} * \sigma(\tilde{r}^{\ell,i}), \tag{10.22}$$

where

$$\tilde{r}^{\ell,i} = r^{\ell,i-1} - A^{\ell,i} * \sigma \circ B^{\ell,i} * r^{\ell,i-1}. \tag{10.23}$$

Proof. First, we apply $A^{\ell,i+1} \circ \sigma \circ B^{\ell,i+1}$ on the both sides of (9.37) and get

$$A^{\ell,i+1} * \sigma \circ B^{\ell,i+1} * r^{\ell,i} = A^{\ell,i+1} * \sigma \circ B^{\ell,i+1} * \sigma(\tilde{r}^{\ell,i}). \tag{10.24}$$

Minus by $r^{\ell,i}$ on the both sides and recall the definition in (10.23), we have

$$\tilde{r}^{\ell,i+1} = r^{\ell,i} - A^{\ell,i+1} * \sigma \circ B^{\ell,i+1} * \sigma(\tilde{r}^{\ell,i}).$$

By the definition of $r^{\ell,i} = \sigma(\tilde{r}^{\ell,i})$, we finish this proof. □

We call the above form (10.22) as σ-ResNet, similar to the MgNet we replace $A^{\ell,i}$ by A^ℓ and get the next Mg-ResNet form as:

$$r^{\ell,i} = \sigma(r^{\ell,i-1}) - A^{\ell} * \sigma \circ B^{\ell,i} * \sigma(r^{\ell,i-1}). \tag{10.25}$$

Table 10.1. Comparison for all iterative forms

Primal-Dual	Model	Iterative form
Feature space	Abstract-MgNet	Solving $A^\ell(u^\ell) = f^\ell$
	General-MgNet	$u^{\ell,i} = u^{\ell,i-1} + B^{\ell,i}(f^\ell - A^\ell(u^{\ell,i-1}))$
	MgNet	$u^{\ell,i} = u^{\ell,i-1} + \sigma \circ B^{\ell,i} \circ \sigma(f^\ell - A^\ell(u^{\ell,i-1}))$
Data space	pre-act ResNet	$r^{\ell,i} = r^{\ell,i-1} - A^{\ell,i} * \sigma \circ B^{\ell,i} * \sigma(r^{\ell,i-1})$
	Mg pre-act ResNet	$r^{\ell,i} = r^{\ell,i-1} - A^{\ell} * \sigma \circ B^{\ell,i} * \sigma(r^{\ell,i-1})$
	Mg-ResNet	$r^{\ell,i} = \sigma(r^{\ell,i-1}) - A^{\ell} * \sigma \circ B^{\ell,i} * \sigma(r^{\ell,i-1})$
	σ-ResNet	$r^{\ell,i} = \sigma(r^{\ell,i-1}) - A^{\ell,i} * \sigma \circ B^{\ell,i} * \sigma(r^{\ell,i-1})$
	ResNet	$r^{\ell,i} = \sigma(f^{\ell,i-1} - A^{\ell,i} * \sigma \circ B^{\ell,i} * r^{\ell,i-1})$

If we take these pooling and prolongation operators as discussed in the previous sections and focus on the iterative forms on a certain grid ℓ, we may compare them all as:

We can have these connections for all iterative scheme in data space:

(10.26)
$$\text{ResNet} \xleftrightarrow{(10.23)} \sigma\text{-ResNet} \xleftrightarrow{A^{\ell,i}\leftrightarrow A^{\ell}} \text{Mg-ResNet} \xleftrightarrow{\sigma(f^{\ell,i-1})\leftrightarrow f^{\ell,i-1}} \text{Mg pre-act ResNet} \xleftrightarrow{A^{\ell}\leftrightarrow A^{\ell,i}} \text{pre-act ResNet}.$$

In this sense, these MgNet related models can be understood as models between pre-act ResNet and ResNet. And all these models can be understood as iteration in the data space as a dual relationship with feature space as MgNet.

The rationality of replacing $A^{\ell,i}$ by layer independent A^ℓ may be justified by the following theorem.

Theorem 25. *On each grid $\mathcal{T}_\ell$,*

1. *Any CNN model with*

$$f^{\ell,i} = \chi^{\ell,i} \circ \sigma(f^{\ell,i-1}), \tag{10.27}$$

can be written as

$$f^{\ell,i} = \sigma(f^{\ell,i-1}) - \xi^\ell \circ \sigma \circ \eta^{\ell,i} \circ \sigma(f^{\ell,i-1}). \tag{10.28}$$

2. *Any CNN model with*

$$f^{\ell,i} = \sigma \circ \chi^{\ell,i}(f^{\ell,i-1}). \tag{10.29}$$

can be written as

$$f^{\ell,i} = \sigma\left(f^{\ell,i-1} - \xi^\ell \circ \sigma \circ \eta^{\ell,i}(f^{\ell,i-1})\right). \tag{10.30}$$

Proof. Let use prove the first case as an example, the second case can be proven with the same process.

With similar structure in MgNet, we can take

$$\xi^\ell = \hat{\delta}^\ell := [\hat{\delta}_1, \cdots, \hat{\delta}_{c_\ell}], \tag{10.31}$$

and

$$\eta^{\ell,i} = [\mathrm{id}_{c_\ell}, -\mathrm{id}_{c_\ell}] \circ (\chi^{\ell,i} - \mathrm{id}_{c_\ell}). \tag{10.32}$$

Here

$$\mathrm{id}_{c_\ell} : \mathbb{R}^{n_\ell \times n_\ell \times c_\ell} \mapsto \mathbb{R}^{n_\ell \times n_\ell \times c_\ell}, \tag{10.33}$$

is the identity map and

$$\hat{\delta}_k : \mathbb{R}^{n_\ell \times n_\ell \times 2c_\ell} \mapsto \mathbb{R}^{n_\ell \times n_\ell}, \tag{10.34}$$

with

$$\hat{\delta}_k([X, Y]) = -([X]_k + [Y]_k), \tag{10.35}$$

for any $X, Y \in \mathbb{R}^{n_\ell \times n_\ell \times c_\ell}$ and $[X, Y] \in \mathbb{R}^{n_\ell \times n_\ell \times 2c_\ell}$.

First, we see that $\eta^{\ell,i}$ with the above form is a convolution from $\mathbb{R}^{n_\ell \times n_\ell \times c_\ell}$ to $\mathbb{R}^{n_\ell \times n_\ell \times 2c_\ell}$. Following the identity

$$ReLU(x) + ReLU(-x) = x, \tag{10.36}$$

and the definition of ξ^ℓ i.e.

$$\xi^\ell = \hat{\delta}^\ell, \tag{10.37}$$

as a special case in MgNet. For more details, we can give a exact form of $\hat{\delta}_k$ as in (10.35) with

$$\hat{\delta}_k = [0, \cdots, 0, -\delta, \cdots 0; 0, \cdots, 0, -\delta, \cdots 0], \quad k = 1 : c_\ell, \tag{10.38}$$

where δ is the identity kernel during one channel.

At last, we have

$$\left[\xi^\ell \circ \sigma \circ [\mathrm{id}_{c_\ell}, -\mathrm{id}_{c_\ell}](x)\right]_k = \left[\xi^\ell \circ \sigma \circ [x, -x]\right]_k, \tag{10.39}$$

$$= \hat{\delta}_k([\sigma(x), \sigma(-x)]), \tag{10.40}$$

$$= -\delta([\sigma(x)]_k) - \delta([\sigma(-x)]_k), \tag{10.41}$$

$$= -(\sigma([x]_k) + \sigma(-[x]_k)), \tag{10.42}$$

$$= -[x]_k \tag{10.43}$$

Thus to say,

$$\xi^\ell \circ \sigma \circ [\mathrm{id}_{c_\ell}, -\mathrm{id}_{c_\ell}] = -\mathrm{id}_{c_\ell}. \tag{10.44}$$

Then the modified dual form of MgNet in (10.22) becomes

$$f^{\ell,i} = \sigma(f^{\ell,i-1}) - \xi^{\ell,i} \circ \sigma \circ \eta^{\ell,i} \circ \sigma(f^{\ell,i-1}), \tag{10.45}$$

$$= \sigma(f^{\ell,i-1}) - \left(\xi^\ell \circ \sigma \circ [\mathrm{id}_{c_\ell}, -\mathrm{id}_{c_\ell}]\right) \circ (\chi^{\ell,i} - \mathrm{id}_{c_\ell}) \circ \sigma(f^{\ell,i-1}) \tag{10.46}$$

$$= \sigma(f^{\ell,i-1}) + (\chi^{\ell,i} - \mathrm{id}_{c_\ell}) \circ \sigma(f^{\ell,i-1}), \tag{10.47}$$

$$= \chi^{\ell,i} \circ \sigma(f^{\ell,i-1}). \tag{10.48}$$

This covers (10.28). □

Remark 12. Theorems 25 shows that general CNN in the forms of either (10.27) or (10.29) can be written recast as (10.28) or (10.30) with the data-feature mapping $A^\ell = \xi^\ell$ that is not only independent of the layers, but is actually given a priori as in (10.31). In view of Theorems 24 and 29, the classic CNN models can be essentially recovered from MgNet by choosing ξ^ℓ a priori as in (10.31). Since the classic CNN models have been extensively tested to be successful, the more general MgNet with more general ξ^ℓ (to be trained) are expected to be more efficient than the classic CNN models.

11

Preconditioned Training Algorithms

11.1 Data normalization in DNNs and CNNs

11.1.1 Normalization for input data of DNNs

Consider that we have the all training data as

$$(X, Y) := \{(x_i, y_i)\}_{i=1}^{N}, \tag{11.1}$$

for $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^k$.

Before we input every data into a DNN model, we will apply the following normalization for all data x_i for each component. Let denote

$$[x_i]_j \longleftrightarrow \text{ the j-th component of data } x_i. \tag{11.2}$$

Then we have following formula of for all $j = 1, 2, \cdots, d$

$$[\tilde{x}_i]_j = \frac{[x_i]_j - [\mu_X]_j}{\sqrt{[\sigma_X]_j}}, \tag{11.3}$$

where

(11.4)

$$[\mu_X]_j = \mathbb{E}_{x\sim X}[[x]_j] = \frac{1}{N}\sum_{i=1}^{N}[x_i]_j, \quad [\sigma_X]_j = \mathbb{V}_{x\sim X}[[x]_j] = \frac{1}{N}\sum_{i=1}^{N}([x_i]_j - [\mu_X]_j)^2.$$

Here $x \sim X$ means that x is a discrete random variable on X with probability

$$\mathbb{P}(x = x_i) = \frac{1}{N}, \tag{11.5}$$

for any $x_i \in X$.

For simplicity, we rewrite the element-wise definition above as the following compact form

$$\tilde{x}_i = \frac{x_i - \mu_X}{\sqrt{\sigma_X}}, \tag{11.6}$$

where

$$x_i, \tilde{x}_i, \mu_X, \sigma_X \in \mathbb{R}^d, \tag{11.7}$$

defined as before and all operations in (11.6) are element-wise.

Here we note that, by normalizing the data set, we have the next properties for new data $\tilde{x} \in \tilde{X}$ with component $j = 1, 2, \cdots, d$,

$$\mathbb{E}_{\tilde{X}}[[\tilde{x}]_j] = \frac{1}{N}\sum_{i=1}^{N}[\tilde{x}_i]_j = 0, \tag{11.8}$$

and

$$\mathbb{V}_{\tilde{X}}[[\tilde{x}]_j] = \frac{1}{N}\sum_{i=1}^{N}([\tilde{x}_i]_j - \mathbb{E}_{\tilde{X}}[[\tilde{x}]_j])^2 = 1. \tag{11.9}$$

Finally, we will have a "new" data set

$$\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_N\}, \tag{11.10}$$

with unchanged label set Y. For the next sections, without special notices, we use X data set as the normalized one as default.

11.1.2 Data normalization for images in CNNs

For images, consider we have a color image data set $(X, Y) := \{(x_i, y_i)\}_{i=1}^N$ where

$$x_i \in \mathbb{R}^{3\times m\times n}. \tag{11.11}$$

We further denote these the (s, t) pixel value for data x_i at channel j as:

$$[x_i]_{j;st} \longleftrightarrow (s, t) \text{ pixel value for } x_i \text{ at channel } j, \tag{11.12}$$

where $1 \le i \le N, 1 \le j \le 3, 1 \le s \le m$, and $1 \le j \le n$.

Then, the normalization for x_i is defined by

$$[\tilde{x}_i]_{j;st} = \frac{[x_i]_{j;st} - [\mu_X]_j}{\sqrt{[\sigma_X]_j}}, \tag{11.13}$$

where

$$[x_i]_{j;st}, [\tilde{x}_i]_{j;st}, [\mu_X]_j, [\sigma_X]_j \in \mathbb{R}. \tag{11.14}$$

Here

$$[\mu_X]_j = \frac{1}{m \times n \times N} \sum_{1 \le i \le N} \sum_{1 \le s \le m, 1 \le t \le n} [x_i]_{j;st}. \tag{11.15}$$

and

$$[\sigma_X]_j = \frac{1}{N \times m \times n} \sum_{1 \le i \le N} \sum_{1 \le s \le m, 1 \le t \le n} ([x_i]_{j;st} - [\mu_X]_j)^2. \tag{11.16}$$

In batch normalization, we confirmed with Lian by both numerical test and code checking that BN also use the above formula to compute the variance in CNN for each channel.

Another way to compute the variance over each channel is to compute the standard deviation on each channel for every data, and then average them in the data direction.

$$\sqrt{[\tilde{\sigma}_X]_j} = \frac{1}{N} \sum_{1 \le i \le N} \left(\frac{1}{m \times n} \sum_{1 \le s \le m, 1 \le t \le n} ([x_i]_{j;st} - [\mu_i]_j)^2 \right)^{\frac{1}{2}}, \tag{11.17}$$

where

$$[\mu_i]_j = \frac{1}{m \times n} \sum_{1 \le s \le m, 1 \le t \le n} [x_i]_{j;st}. \tag{11.18}$$

11.1.3 Comparison of $\sqrt{[\sigma_X]_j}$ and $\sqrt{[\tilde{\sigma}_X]_j}$ on CIFAR10.

They share the same μ_X as

$$\mu_X = \Big(0.49140105\ 0.48215663\ 0.44653168\Big). \tag{11.19}$$

But they had different standard deviation estimates:

$$\begin{aligned} \sqrt{[\sigma_X]_j} &= \Big(0.24703284\ 0.24348499\ 0.26158834\Big) \\ \sqrt{[\tilde{\sigma}_X]_j} &= \Big(0.20220193\ 0.19931635\ 0.20086373\Big) \end{aligned} \tag{11.20}$$

11.2 Initialization for deep neural networks

11.2.1 Xavier's Initialization

The goal of Xavier initialization [4] is to initialize the deep neural network to avoid gradient vanishing or blowup when the input is white noise.

Let us denote the DNN models as:

$$\begin{cases} f^1(x) &= W^1 x + b^1 \\ f^\ell(x) &= W^\ell \sigma(f^{\ell-1}(x)) + b^\ell \quad \ell = 2:L\,, \\ f(x) &= f^L \end{cases} \tag{11.21}$$

with $x \in \mathbb{R}^{n_0}$ and $f^\ell \in \mathbb{R}^{n_\ell}$. More precisely, we have

$$W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}. \tag{11.22}$$

The basic assumptions that we make are:

- The initial weights W^ℓ_{ij} are i.i.d symmetric random variables with mean 0, namely the probability density function of W^ℓ_{ij} is even.
- The initial bias $b^\ell = 0$.

Now we choose the variance of the initial weights to ensure that the features f^L and gradients don't blow up or vanish. To this end we have the following lemma.

Lemma 30. *Under the previous assumptions f^ℓ_i is a symmetric random variable with $\mathbb{E}[f^\ell] = 0$. Moreover, we have the following identity*

$$\mathbb{E}[(f^\ell_i)^2] = \sum_k \mathbb{E}[(W^\ell_{ik})^2]\mathbb{E}[\sigma(f^{\ell-1}_k)^2]. \tag{11.23}$$

Proof. For general activation function $\sigma(x)$, let first prove that f^ℓ_i is a symmetric random variable with $\mathbb{E}[f^\ell] = 0$. For any ℓ and $1 \le i \le n_\ell$, we have

$$f^\ell_i = \sum_k W^\ell_{ik}\sigma(f^{\ell-1}_k) + b_i, \tag{11.24}$$

Taking expectations, noting that W^ℓ_{ik} are independent of $f^{\ell-1}_k$ and that $b_i = 0$, we get

$$\mathbb{E}[f^\ell_i] = \sum_k \mathbb{E}[W^\ell_{ik}]\mathbb{E}[\sigma(f^{\ell-1}_k)] = 0. \tag{11.25}$$

since $\mathbb{E}[W^\ell_{ik}] = 0$. Moreover, if the W^ℓ_{ij} are symmetric, it is clear that f^ℓ_i will be.

Next we calculate the variance of f^ℓ_i, we get

$$\begin{aligned} \mathbb{V}[f^\ell_i] = \mathbb{E}[(f^\ell_i)^2] &= \mathbb{E}\left[\left(\sum_k W^\ell_{ik}\sigma(f^{\ell-1}_k)\right)\left(\sum_l W^\ell_{il}\sigma(f^{\ell-1}_l)\right)\right] \\ &= \sum_{k,l} \mathbb{E}[W^\ell_{ik} W^\ell_{il}\sigma(f^{\ell-1}_k)\sigma(f^{\ell-1}_l)]. \end{aligned} \tag{11.26}$$

In the above sum we will have W_{ik}^{ℓ} and W_{il}^{ℓ} independent unless $k = l$. In this case the term will be 0. So the only terms which remain are those for which $k = l$ and we get

$$\mathbb{E}[(f_i^{\ell})^2] = \sum_k \mathbb{E}[(W_{ik}^{\ell})^2]\mathbb{E}[\sigma(f_k^{\ell-1})^2]. \tag{11.27}$$

□

Now, if $\sigma = id$,let us do this by induction from $\ell = 1$

- $\ell = 1$
  Let us first think about $\ell = 1$, we will have

$$\mathbb{E}[(f_i^1)^2] = \sum_k \mathbb{E}[(W_{ik}^1)^2]\mathbb{E}[([x]_k)^2] = \mathbb{E}[(W_{ik}^1)^2]\left(\sum_k \mathbb{E}[([x]_k)^2]\right), \tag{11.28}$$

  for any $i = 1, 2, \cdots, n_1$.
- $\ell = 2$
  Because we already know that

$$\mathbb{V}[f_i^1] = \mathbb{E}[(W_{ik}^1)^2]\left(\sum_k \mathbb{E}[([x]_k)^2]\right) = \mathbb{V}[f_j^1], \quad \forall i, j, \tag{11.29}$$

  from derivation in $\ell = 1$. We will see that

$$\mathbb{V}[f_i^2] = n_1\mathbb{V}[W_{ik}^2]\mathbb{V}[f_k^1] = n_1\mathbb{V}[W_{st}^2]\left(\mathbb{V}[W_{st}^1]\sum_k \mathbb{E}[([x]_k)^2]\right). \tag{11.30}$$

- $\ell = \cdots$
- $\ell = L$

$$\mathbb{V}[f_i^L] = \left(\Pi_{\ell=2}^L n_{\ell-1}\mathrm{Var}[W_{st}^{\ell}]\right)\left(\mathbb{V}[W_{st}^1]\sum_k \mathbb{E}[([x]_k)^2]\right). \tag{11.31}$$

We make this assumption that $\sigma = id$, which is pretty reasonably since most activation functions in use at the time (such as the hyperbolic tangent) were close to the identity near 0.

Now, if we set

$$\mathbb{V}[W_{ik}^{\ell}] = \frac{1}{n_{\ell-1}}, \quad \forall \ell \geq 2, \tag{11.32}$$

we will obtain

$$\mathbb{V}[f_i^L] = \mathbb{V}[f_j^{L-1}] = \cdots = \mathbb{V}[f_k^1] = \mathbb{V}[W_{st}^1]\sum_k \mathbb{E}[([x]_k)^2]. \tag{11.33}$$

Thus, in pure DNN models, it is enough to just control $\sum_k \mathbb{E}[([x]_k)^2]$.

A similar analysis of the propagation of the gradient ($\frac{\partial L(\theta)}{\partial f^{\ell}}$) suggests that we set

$$\mathbb{V}[W_{ik}^{\ell}] = \frac{1}{n_{\ell}}. \tag{11.34}$$

Thus, the **Xavier's initialization** suggests to initialize W_{ik}^{ℓ} with variance as:

- To control $\mathbb{V}[f_i^\ell]$:

$$\mathrm{Var}[W_{ik}^\ell] = \frac{1}{n_{\ell-1}}. \tag{11.35}$$

- To control $\mathbb{V}[\frac{\partial L(\theta)}{\partial f_i^\ell}]$:

$$\mathrm{Var}[W_{ik}^\ell] = \frac{1}{n_\ell}. \tag{11.36}$$

- Trade-off to control $\mathbb{V}[\frac{\partial L(\theta)}{\partial W_{ik}^\ell}]$:

$$\mathrm{Var}[W_{ik}^\ell] = \frac{2}{n_{\ell-1}+n_\ell}. \tag{11.37}$$

Here we note that, this analysis works for all symmetric type distribution around zero, but we often just choose uniform distribution $\mathcal{U}(-a,a)$ and normal distribution $\mathcal{N}(0,s^2)$. Thus, the final version of Xavier's initialization takes the trade-off type as

$$W_{ik}^\ell \sim \mathcal{U}(-\sqrt{\frac{6}{n_\ell+n_{\ell-1}}}, \sqrt{\frac{6}{n_\ell+n_{\ell-1}}}), \tag{11.38}$$

or

$$W_{ik}^\ell \sim \mathcal{N}(0, \frac{2}{n_\ell+n_{\ell-1}}). \tag{11.39}$$

11.2.2 Variance analysis in backward propagation phase

Recall the loss function

$$L(\theta) = \mathbb{E}_{(x,y)\sim D}\ell\left(\mathrm{softmax}\left(f^L\right), y\right)$$

Where $f^L(x;\theta)$ is the DNN function defined by

$$\begin{cases} f^1(x) = W^1x + b^1 \\ f^l(x) = W^l\sigma\left(f^{l-1}(x)\right) + b^l \quad \forall l = 2, \cdots L \end{cases}$$

Then We have the following "BP" formula

$$\frac{\partial L(\theta)}{\partial W^l} = \frac{\partial L}{\partial f^l}\cdot\frac{\partial f^l}{\partial W^l}$$

where $\frac{\partial L}{\partial f^l} \in \mathbb{R}^{n_e}$, $\frac{\partial f}{\partial W^l} \in \mathbb{R}^{n_e\times(n_e\times n_{e-1})}$. More Precisely,

$$\frac{\partial L(\theta)}{\partial W_{st}^l} = \sum_{i=1}^{n_l}\left(\frac{\partial L(\theta)}{\partial f_i^l}\cdot\frac{\partial f_i^l}{\partial W_{st}^l}\right).$$

Assume:

1. $\left\{\frac{\partial L(\theta)}{\partial f_i^l}\cdot\frac{\partial f_i^l}{\partial W_{st}^L}\right\}_{i=1}^{n_e}$ are independent
2. For each i, $\frac{\partial L(\theta)}{\partial f_i^l}$ & $\frac{\partial f_i^l}{\partial W_{st}^l}$ are independent

Actually, we can not make general assumption for this as $L(\theta), f_i^l, \frac{\partial f_i^l}{\partial W_{st}}\frac{\partial L(\theta)}{\partial f_i^l}$ are all fixed.

Then We have,

$$\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^l}\right]=\sum_{i=1}^{n_l}\left(\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_i^l}\right)^2\right]\mathbb{E}\left[\left(\frac{\partial f_i^l}{\partial W_{st}^l}\right)^2\right]-\left(\mathbb{E}\left[\frac{\partial L(\theta)}{\partial f_i^l}\right]\mathbb{E}\left[\frac{\partial f_i^l}{\partial W_{st}^l}\right]\right)^2\right)$$

- Consider $\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_i^l}\right)^2\right]$. By chain rule:

$$\begin{aligned}\frac{\partial L(\theta)}{\partial f_i^l}&=\frac{\partial L(\theta)}{\partial f^{l+1}}\cdot\frac{\partial f^{l+1}}{\partial f_i^l}\\&=\sum_{j=1}^{n_{l+1}}\frac{\partial L(\theta)}{\partial f_j^{l+1}}\frac{\partial f_j^{l+1}}{\partial f_i^l}\\&=\sum_{j=1}^{n_{l+1}}W_{ji}^{l+1}\sigma'\left(f_i^l\right)\frac{\partial L(\theta)}{\partial f_j^{l+1}}\\&=\sum_{j=1}^{n_{l+1}}W_{ji}^{l+1}\frac{\partial L(\theta)}{\partial f_j^{l+1}}(\text{ if }\sigma=id)\end{aligned}$$

  Keep doing this:

$$\frac{\partial L(\theta)}{\partial f^l}=\left[W^{l+1}\right]^\top\cdot\left[W^{l+2}\right]^\top\cdots\left[W^L\right]^T\frac{\partial L(\theta)}{\partial f^L}$$

  Assume the independent of $\frac{\partial L(\theta)}{\partial f^L}$ with $W^{l+i}, i=1,2,\ldots$ Can not be ture, as $\frac{\partial L(\theta)}{\partial f^l}$ contains W^{l+i}

$$\mathbb{E}\left[\frac{\partial L(\theta)}{\partial f_i^l}\right]=\sum_j\mathbb{E}\left[W_{ji}^{l+1}\right]\mathbb{E}\left[\frac{\partial L(\theta)}{\partial f_j^l}\right]=0$$

$$\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_i^l}\right)^2\right]=\mathbb{V}\left[\frac{\partial L(\theta)}{\partial f_i^l}\right]=\prod_{j=l+1}^{L-1}n_j\mathbb{V}\left[W_{st}^j\right]\cdot\left(\mathbb{V}\left[W_{st}^L\right]\sum_{i=1}^{n_L}\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_i^L}\right)^2\right]\right)$$

- Consider $\mathbb{E}\left[\left(\frac{\partial f_i^l}{\partial W_{st}^l}\right)^2\right]$. By definition,

$$\frac{\partial f_i^l}{\partial W_{st}^l}=\delta_{is}\sigma\left(f_t^{l-1}\right)$$

  namely, only $\frac{\partial f_s^l}{\partial W_{st}^l}\neq 0$, and $\frac{\partial f_s^l}{\partial W_{st}^l}=\sigma\left(f_t^{l-1}\right)$
  If σ=id, apply the forward results

1. Expectation:

$$\mathbb{E}\left[\left(\frac{\partial f_s^l}{\partial W_{st}^l}\right)\right] = 0.$$

2. Variance:

$$\begin{aligned}\mathbb{E}\left[\left(\frac{\partial f_s^l}{\partial W_{st}^l}\right)^2\right] &= \mathbb{E}\left[\left(f_t^{l-1}\right)^2\right] = \mathbb{V}\left[f_t^{l-1}\right] \\ &= \prod_{j=2}^{l-1} n_{j-1}\mathbb{V}\left[W_{st}^j\right]\cdot\left(\mathbb{V}\left[W_{st}^1\right]\sum_{k=1}^{d}\mathbb{E}\left[X_k^2\right]\right).\end{aligned} \tag{11.40}$$

- Finally, We have:

$$\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^l}\right] = \prod_{j=l+1}^{L-1} n_j\mathbb{V}\left[W_{st}^j\right]\left(\mathbb{V}\left[W_{st}^L\right]\sum_{k=1}^{n_L}\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_k^2}\right)^2\right]\right)\times\prod_{j=2}^{l-1} n_j\mathbb{V}\left[W_{st}^j\right]\left(\mathbb{V}\left[W_{st}^1\right]\sum_{R=1}^{d}\mathbb{E}\left[X_k^2\right]\right)$$

- Thus, we may consider

$$\mathbb{V}\left[W_{st}^l\right] = f\left(n_l, n_{l-1}\right)$$

Summary

- Control $\mathbb{V}\left[f_i^l\right]$:

$$\begin{aligned}&\prod_{j=2}^{l} n_{j-1}\mathbb{V}\left[W_{st}^j\right]\left(\mathbb{V}\left[W_{st}^1\right]\sum_{k=1}^{d}\mathbb{E}\left[X_k^2\right]\right) \\ &\Rightarrow \mathbb{V}\left[W_{st}^l\right] = \frac{1}{n_{l-1}}\end{aligned}$$

- Control $\mathbb{V}\left[\frac{\partial L(\theta)}{\partial f_i^2}\right]$:

$$\begin{aligned}\mathbb{V}\left[\frac{\partial L(\theta)}{\partial f_i^2}\right] &= \prod_{j=1+1}^{L-1} n_j\mathbb{V}\left[W_{st}^j\right]\cdot\left(\mathbb{V}\left[W_{st}^L\right]\sum_{i=1}^{n_L}\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_i^l}\right)^2\right]\right) \\ &\Rightarrow \mathbb{V}\left[W_{st}^l\right] = \frac{1}{n_l}\end{aligned}$$

- Control $\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^l}\right]$:

$$\begin{aligned}&\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^l}\right] = \mathbb{V}\left[\frac{\partial L(\theta)}{\partial f_s^l}\right]\cdot\mathbb{V}\left[f_t^{l-1}\right] \\ &= \prod_{j=l+1}^{L-1} n_j\mathbb{V}\left[W_{st}^j\right]\left(\mathbb{V}\left[W_{st}^L\right]\sum_{k=1}^{n_L}\mathbb{E}\left[\left(\frac{\partial L(\theta)}{\partial f_k^L}\right)^2\right]\right)\times\prod_{j=2}^{l-1} n_j V\left[W_{st}^j\right]\left(\mathbb{V}\left[W_{st}^1\right]\sum_{R=1}^{d}\mathbb{E}\left[X_k^2\right]\right) \\ &= \prod_{j=l+1}^{L-1} n_j\cdot f\left(n_j, n_{j-1}\right)\cdot\prod_{j=2}^{l-1} n_{j-1} f\left(n_j, n_{j-1}\right)\times\mathbb{V}_1\times\mathbb{V}_L\end{aligned}$$

Where $\mathbb{V}\left[W_{st}^{j}\right] = f\left(n_j, n_{j-1}\right), \mathbb{V}_1 = \left(\mathbb{V}\left[W_{st}^{1}\right]\sum_{k=1}^{d}\mathbb{E}\left[X_k^2\right]\right), \mathbb{V}_L = \mathbb{V}\left[W_{st}^{L}\right]\sum_{k=1}^{n_L}\mathbb{E}\left[(\frac{\partial L(\theta)}{\partial f_k^L})^2\right]$.

- 1. If $f\left(n_1, n_{j-1}\right) = \frac{1}{n_{j-1}}$ (control $W[f_i^l]$),

$$\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^{l}}\right] = \prod_{j=l+1}^{L-1}\frac{n_j}{n_{j-1}} \times \mathbb{V}_1 \times \mathbb{V}_2$$

   This Will decrease as l groW up. (Notice that $n_l > n_k$ if $l > k$)
2. If $f\left(n_j, n_{j-1}\right) = \frac{1}{n_j}$ (control $V[\frac{\partial L(\theta)}{\partial f_i^L}]$),

$$\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^{l}}\right] = \prod_{j=2}^{l-1}\frac{n_{j-1}}{n_j} \times \mathbb{V}_1 \times \mathbb{V}_2$$

   Still Decrease!
3. If $f\left(n_j, n_{j-1}\right) = \frac{2}{n_j+n_{j-1}}(\leq \frac{1}{\sqrt{n_j n_{j-1}}})$,

$$\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^{l}}\right] = \frac{\sqrt{n_{L-1}}\cdot\sqrt{n_1}}{\sqrt{n_l}\cdot\sqrt{n_{l-1}}} \times \mathbb{V}_1 \times \mathbb{V}_L$$

   Decrease!

Question

Can we design some other choices of $\mathbb{V}\left[W_{st}^{l}\right]$ such that $\mathbb{V}\left[\frac{\partial L(\theta)}{\partial W_{st}^{l}}\right]$ will

- Keep constant ?
- Increase ?
- Change with certain scale ?

11.2.3 Kaiming's initialization

In [6], Kaiming He and others extended this analysis to get an *exact* result when the activation function is the **ReLU**.

We first have the following lemma for symmetric distribution.

Lemma 31. *If $X_i \in \mathbb{R}$ for $i = 1 : n$ are i.i.d with symmetric probability density function $p(x)$, i.e. $p(x)$ is even. Then for any nonzero random vector $Y = (Y_1, Y_2, \cdots, Y_n) \in \mathbb{R}^n$ which is independent with X_i, the following random variable*

$$Z = \sum_{i=1}^{n} X_i Y_i, \tag{11.41}$$

is also symmetric.

Proof. Let us denote the joint distribution function for Y as $q(y_1, \cdots, y_n)$. Then the joint distribution density function for (X, Y) is

$$f(x_1, \cdots, x_n, y_1, \cdots, y_n) = p(x_1)p(x_2)\cdots p(x_n)q(y_1, y_2, \cdots, y_n). \tag{11.42}$$

Then we have the following probability function

$$\begin{aligned}
\mathbb{P}(Z < z) &= \int_{\sum_{i=1}^n x_i y_i < z} f(x_1, \cdots, x_n, y_1, \cdots, y_n) dxdy \\
&= \int_{\sum_{i=1}^n x_i y_i < z} p(x_1)p(x_2)\cdots p(x_n)q(y_1, y_2, \cdots, y_n) dxdy \\
&= \int_{\sum_{i=1}^n -x_i y_i > -z} p(x_1)p(x_2)\cdots p(x_n)q(y_1, y_2, \cdots, y_n) dxdy \\
&= \int_{\sum_{i=1}^n -x_i y_i > -z} p(-x_1)p(-x_2)\cdots p(-x_n)q(y_1, y_2, \cdots, y_n) dxdy \\
&= \int_{\sum_{i=1}^n \tilde{x}_i y_i > -z} p(\tilde{x}_1)p(\tilde{x}_2)\cdots p(\tilde{x}_n)q(y_1, y_2, \cdots, y_n) d\tilde{x}dy \\
&= \mathbb{P}(Z > -z).
\end{aligned} \tag{11.43}$$

□

Then state the following result for ReLU function and random variable with symmetric distribution around 0.

Lemma 32. *If X is a random variable on $\mathbb{R}$ with symmetric probability density $p(x)$ around zero, i.e.,*

$$p(x) = p(-x). \tag{11.44}$$

Then we have $\mathbb{E}X = 0$ and

$$\mathbb{E}[[\mathrm{ReLU}(X)]^2] = \frac{1}{2}\mathrm{Var}[X]. \tag{11.45}$$

Proof. By definition

$$\mathbb{E}X = \int_{-\infty}^{\infty} xp(x)d(x) = 0. \quad (\text{since } p(x) = p(-x)). \tag{11.46}$$

Furthermore,

$$\begin{aligned}
\mathbb{E}[[\mathrm{ReLU}(X)]^2] &= \int_{-\infty}^{\infty} (\mathrm{ReLU}(x))^2 p(x)dx \\
&= \int_0^{\infty} x^2 p(x)dx \\
&= \frac{1}{2}\int_{-\infty}^{\infty} (x - \mathbb{E}[x])^2 p(x)dx \\
&= \frac{1}{2}\mathbb{E}[[X - \mathbb{E}X]^2] \\
&= \frac{1}{2}\mathbb{V}[X].
\end{aligned} \tag{11.47}$$

□

Based on the previous Lemma 30, we know that $f_k^{\ell-1}$ is a symmetric distribution around 0. The most important observation in Kaiming's paper [6] is that:

$$\mathbb{V}[f_i^\ell] = n_{\ell-1}\mathbb{V}[W_{ij}^\ell]\mathbb{E}[[\sigma(f_j^{\ell-1})]^2] = n_{\ell-1}\mathbb{V}[W_{ik}^\ell]\frac{1}{2}\mathbb{V}[f_k^{\ell-1}], \tag{11.48}$$

if σ = ReLU. Thus, Kaiming's initialization suggests to take:

$$\mathbb{V}[W_{ik}^\ell] = \frac{2}{n_{\ell-1}},, \quad \forall \ell \geq 2. \tag{11.49}$$

For the first layer $\ell = 1$, by definition

$$f^1 = W^1 x + b^1, \tag{11.50}$$

there is no ReLU, thus it should be $\mathbb{V}[W_{ik}^1] = \frac{1}{d}$. For simplicity, they still use $\mathbb{V}[W_{ik}^1] = \frac{2}{d}$ in the paper [6].

Similarly, an analysis of the propagation of the gradient suggests that we set

$$\mathbb{V}[W_{ik}^\ell] = \frac{2}{n_\ell}. \tag{11.51}$$

However, in paper [6] authors did not suggest to take the trade-off version, they just chose

$$\mathbb{V}[W_{ik}^\ell] = \frac{2}{n_{\ell-1}}, \tag{11.52}$$

as default.

Thus, the final version of Kaiming's initialization takes the forward type as

$$W_{ik}^\ell \sim \mathcal{U}(-\sqrt{\frac{6}{n_{\ell-1}}}, \sqrt{\frac{6}{n_{\ell-1}}}), \tag{11.53}$$

or

$$W_{ik}^\ell \sim \mathcal{N}(0, \frac{2}{n_{\ell-1}}). \tag{11.54}$$

11.2.4 Initialization in CNN models and experiments

For CNN models, following the analysis above we have the next iterative scheme in CNNs

$$f^{\ell,i} = K^{\ell,i} * \sigma(f^{\ell,i-1}), \tag{11.55}$$

where $f^{\ell,i-1} \in \mathbb{R}^{c_\ell \times n_\ell \times m_\ell}$, $f^{\ell,i} \in \mathbb{R}^{h_\ell \times n_\ell \times m_\ell}$ and $K \in \mathbb{R}^{(2k+1)\times(2k+1)\times h_\ell \times c_\ell}$. Thus we have

$$[f^{\ell,i}]_{h;p,q} = \sum_{c=1}^{c_\ell} \sum_{s,t=-k}^{k} K^{\ell,i}_{h,c;s,t} * \sigma([f^{\ell,i-1}]_{c;p+s,q+t}). \tag{11.56}$$

Take variance on both sides, we will get

$$\mathbb{V}[[f^{\ell,i}]_{h;p,q}] = c_\ell(2k+1)^2 \mathbb{V}[K^{\ell,i}_{h,o;s,t}] \mathbb{E}[([f^{\ell,i-1}]_{o;p+s,q+t})^2], \tag{11.57}$$

thus we have the following initialization strategies:

Xavier's initialization

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{2}{(c_\ell + h_\ell)(2k+1)^2}. \tag{11.58}$$

Kaiming's initialization

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{2}{c_\ell(2k+1)^2}. \tag{11.59}$$

Here we can take this Kaiming's initialization as:

- Double the Xavier's choice, and get

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{4}{(c_\ell + h_\ell)(2k+1)^2}. \tag{11.60}$$

- Then pick c_ℓ or h_ℓ for final result

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{4}{(c_\ell + h_\ell)(2k+1)^2} = \frac{2}{c_\ell(2k+1)^2}. \tag{11.61}$$

And they have the both uniform and normal distribution type.

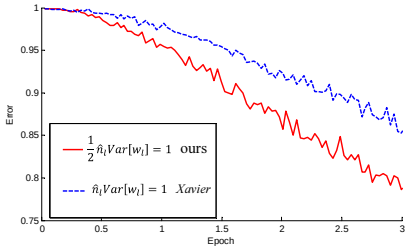


Fig. 11.1. The convergence of a **22-layer** large model. The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. Use ReLU as the activation for both cases. Both Kaiming's initialization (red) and "*Xavier's*" (blue) [4] lead to convergence, but Kaiming's initialization starts reducing error earlier.

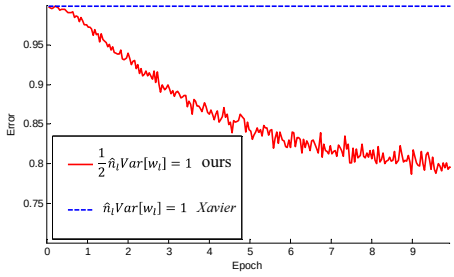


Fig. 11.2. The convergence of a **30-layer** small model (see the main text). Use ReLU as the activation for both cases. Kaiming's initialization (red) is able to make it converge. But "*Xavier's*" (blue) [4] completely stalls - It is also verified that that its gradients are all diminishing. It does not converge even given more epochs.

Given a 22-layer model, in cifar10 the convergence with Kaiming's initialization is faster than Xavier's, but both of them are able to converge and the validation accuracies with two different initialization are about the same(error is 33.82,33.90).

With extremely deep model with up to 30 layers, Kaiming's initialization is able to make the model convergence. On the contrary, Xavier's method completely stalls the learning.

11.3 Batch Normalization in DNN and CNN

11.3.1 Recall the original DNN model

Consider the classical (fully connected) artificial deep neural network (DNN) f^L,

$$\begin{cases} f^1 &= \theta^1(x) := W^1 x + b^1, \\ f^\ell &= \theta^\ell \circ \sigma(f^{\ell-1}) := W^\ell \sigma(f^{\ell-1}) + b^\ell, \ \ell = 2, ..., L. \end{cases} \tag{11.62}$$

where $x \in \mathbb{R}^n$ is the input vector, σ is a non-linear function (activation).

Denote

$$\Theta = (W^1, b^1, \cdots, W^L, b^L), \tag{11.63}$$

$$\theta^\ell = (W^\ell, b^\ell) \in \mathbb{R}^{n_\ell \times (n_{\ell-1}+1)}, \tag{11.64}$$

$$\theta^\ell(x) = W^\ell x + b^\ell. \tag{11.65}$$

We can define

$$\mathrm{DNN_L} := \{\mathrm{f^L(x;\Theta)} \mid \Theta \in \mathbb{R}^N\}. \tag{11.66}$$

Or, we have a more comprehensive notation for classical DNN models.

$$\begin{aligned} \mathrm{DNN_L} := \{f : f = \theta^L \circ \sigma \circ \theta^{L-1} \cdots \sigma \circ \theta^1(x), \\ \theta^\ell \in \mathbb{R}^{n^{\ell+1} \times (n^\ell+1)}, \quad n^0 = d, \quad n^L = 1, \quad n^\ell \in \mathbb{N}^+\}. \end{aligned} \tag{11.67}$$

Generally speaking, a deep learning problem consists of the next few components:

$$\text{Data} \to \text{Model} \to \text{Training} \to \text{Testing}. \tag{11.68}$$

We will try to give a different explanation for batch normalization based on the above outline.

11.3.2 Ideas Behind the BN for DNN: Internal Covariate Shift in Training

The Internal Covariate Shift is defined in [9] as the change in the distribution of network activations due to the change in network parameters during training.

For example, considering the training data with $X = \{(x_i, y_i) \mid i = 1, .., N\} \subset \mathcal{X} := \{(x, y) \mid x, \in \mathbb{R}^n, y \in \mathbb{R}^c\}$, then out put of i-th layer (the input of activation function in $i+1$-the layer) $\{f^i(x_j)\}_{j=1}^N$ will satisfy different discrete distributions if you have different parameters Θ which is always happened during training process.

More intuitively, Fixed distribution of inputs to a sub-network would have positive consequences for the layers *outside* the sub-network, as well. Consider a layer with a sigmoid activation function $f = \sigma(Wx + b)$ where x is the layer input, the weight matrix W and bias vector b are the layer parameters to be learned, and $\sigma(x) = \frac{1}{1+\exp(-x)}$. It is easy to see

$$\sigma'(x) \to 0 \quad \text{if} \quad |x| \to \infty.$$

This means that for all dimensions of $y = Wx+x$ except those with small absolute values, the gradient flowing down to x will vanish and the model will train slowly. However, since y is affected by W, b and the parameters of all the layers below, changes to those parameters during training will likely move many dimensions of y into the saturated regime of the nonlinearity and slow down the convergence. This effect is amplified as the network depth increases. In practice, the saturation problem and the resulting vanishing gradients are usually addressed by using Rectified Linear Units $\mathrm{ReLU}(x) = \max(x, 0)$, careful initialization and small learning rates. However, if we could ensure that the distribution of nonlinearity inputs remains more stable as the network trains, then the optimizer would be less likely to get stuck in the saturated regime, and the training would accelerate.

As a summary: vanishing of gradient:

- Saturated problem:

$$\sigma'(x) \to 0 \quad \text{for } x \text{ localed in some saturated domain.}$$

- Instability from multiplication:

$$\nabla_{W^1} \ell(f^L(x_i; \Theta), y_i) \approx \Pi_{\ell=1}^{L} W^\ell \mathrm{Diag}(\sigma'(f^{\ell-1})) \to 0\,, \quad \text{if } L \text{ is big.}$$

Under this principle, it has been long known [13, 23] that the network training converges faster if its inputs are whitened – i.e., linearly transformed to have zero means and unit variances, and decorrelated.

Thus to say, to improve the training, one may seek to reduce the internal covariate shift. By fixing the distribution of the layer inputs f^ℓ as the training progresses, it is expected to improve the training speed. As each layer observes the inputs produced by the layers below, it would be advantageous to achieve the same whitening of the inputs of each layer. By whitening the inputs to each layer, BN would take a step towards achieving the fixed distributions of inputs that would remove the ill effects of the internal covariate shift.

Within this framework, whitening the layer inputs is expensive, as it requires computing the covariance matrix

$$\mathrm{Cov}[f^\ell] = \mathbb{E}_{x\in X}[f^\ell(x)[f^\ell(x)]^T] - \mathbb{E}_{x\in X}[f^\ell(x)]\mathbb{E}_{x\in X}[f^\ell(x)]^T \tag{11.69}$$

and its inverse square root, to produce the whitened activations

$$\tilde{f}^\ell = \left(\mathrm{Cov}[f^\ell]\right)^{-1/2}\left(f^\ell - \mathbb{E}[f^\ell]\right), \tag{11.70}$$

as well as the derivatives of these transforms for backpropagation. However, this is impossible to take gradient for $\left(\mathrm{Cov}[f^\ell]\right)^{-1/2}$ w.r.t Θ as $f^\ell = f^\ell(x; \Theta)$.

The original version is to do whitening in Batch i.e. $\mathbb{E}_{x\in X}[\cdot]$ that is why this is called Batch Normalization not mini-batch normalization which is the practical version of batch normalization.

11.3.3 Practical batch normalization: assume i.i.d and add scale and shift

Take BN for each scalar feature (neuron).

Since the full whitening of each layer's inputs is costly and not everywhere differentiable, BN makes two necessary simplifications. The first is that instead of whitening the features in layer inputs and outputs **jointly**, BN will normalize each **scalar feature (neuron)** independently, by making it have the mean of zero and the variance of 1. In some sense, this it looks like to add and assumption for the distribution of scalar features to be i.i.d. For a layer with n^ℓ-dimensional input $f^\ell = ([f^\ell]_1, \ldots, [f^\ell]_{n^\ell})$, we will normalize each dimension

$$[\hat{f}^\ell]_k = \frac{[f^\ell]_k - \mathbb{E}[[f^\ell]_k]}{\sqrt{\mathrm{Var}[[f^\ell]_k]}}$$

where the expectation and variance are computed over the training data set. As shown in [13], such normalization speeds up convergence, even when the features are not decorrelated.

Add scale and shift into BN.

Note that simply normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. To address this, we make sure that *the transformation inserted in the network can represent the identity transform*. To accomplish this, we introduce, for each activation $\hat{f}_k^\ell$, a pair of parameters $\gamma_k^\ell, \beta_k^\ell$, which scale and shift the normalized value:

$$[\tilde{f}^\ell]_k = \gamma_k^\ell [\hat{f}^\ell]_k + \beta_k^\ell. \tag{11.71}$$

These parameters are learned along with the original model parameters, and restore the representation power of the network. Indeed, by setting $\gamma_k^\ell = \sqrt{\mathrm{Var}[[f^\ell]_k]}$ and $\beta_k^\ell = \mathbb{E}[[f^\ell]_k]$, we could recover the original activations, if that were the optimal thing to do.

11.3.4 Model with batch normalization

Definition of BN operation based on the batch

Following the idea in (11.68), we consider that we have the all training data as

$$(X, Y) := \{x_i, y_i\}_{i=1}^N. \tag{11.72}$$

Since the normalization is applied to each activation independently, let us focus on a particular activation $[f^\ell]_k$ and omit k as f^ℓ for clarity. We have N values of this activation in the batch,

$$X = \{x_1, \cdots, x_N\}.$$

Let the normalized values be $\hat{f}^\ell$, and their linear transformations be $\tilde{f}^\ell$.
(11.73)

$$\mu_X^\ell \leftarrow \mathbb{E}_{x\sim X}[f^\ell(x)] = \frac{1}{N}\sum_{i=1}^N f^\ell(x_i) \qquad \text{batch mean}$$

$$\sigma_X^\ell \leftarrow \mathbb{E}_{x\sim X}\left[(f^\ell(x) - \mathbb{E}_{x\sim X}[f^\ell(x)])^2\right] = \frac{1}{N}\sum_{i=1}^N (f^\ell(x_i) - \mu_X)^2 \qquad \text{batch variance}$$

$$\hat{f}^\ell(x) \leftarrow \frac{f^\ell(x) - \mu_X^\ell}{\sqrt{\sigma_X^\ell + \epsilon}} \qquad \text{normalize}$$

$$\tilde{f}^\ell(x) \leftarrow \gamma^\ell \hat{f}^\ell(x) + \beta^\ell \qquad \text{scale and shift}$$

Here we note that all these operations in the previous equation are defined by element-wise. Then at last, we define the BN operation based on the batch set as

$$\mathrm{BN}_X(f^\ell(x)) = \tilde{f}^\ell(x) := \gamma^\ell \frac{f^\ell(x) - \mu_X^\ell}{\sqrt{\sigma_X^\ell + \epsilon}} + \beta^\ell, \tag{11.74}$$

where $\tilde{f}^\ell(x)$, μ_X^ℓ and σ_X^ℓ are given above.

Model with BN

In summary, we have the new DNN model with BN as:

$$\begin{cases} \tilde{f}^1(x_i) &= (\theta^1(x_i)), \\ \tilde{f}^\ell &= \theta^\ell \circ \sigma \circ \mathrm{BN}_X(\tilde{f}^{\ell-1}), \quad \ell = 2, ..., L. \end{cases} \tag{11.75}$$

For a more comprehensive notation, we can use the next notation

$$\sigma_{\mathrm{BN}} := \sigma \circ \mathrm{BN}_X. \tag{11.76}$$

Here one thing is important that we need to mention is that because of the new scale γ^ℓ and shift β^ℓ added after the BN operation. We can remove the basis b^ℓ in θ^ℓ, thus to say the real model we will compute should be

$$\begin{cases} \tilde{f}^1(x_i) &= W^1 x_i, \\ \tilde{f}^\ell &= W^\ell \sigma_{\mathrm{BN}}(\tilde{f}^{\ell-1}), \quad \ell = 2, ..., L. \end{cases} \tag{11.77}$$

Combine the two definition, we note

$$\tilde{\Theta} := \{W, \gamma, \beta\}, \tag{11.78}$$

where $W = \{W^1, \cdots, W^l\}$, $\gamma := \{\gamma^2, \cdots, \gamma^L\}$ and $\beta := \{\beta^2, \cdots, \beta^L\}$.

Finally, we have the loss function as:

$$\mathcal{L}(\tilde{\Theta}) = \mathbb{E}_{(x,y)\sim(X,Y)} \approx \frac{1}{N}\sum_{i=1}^{N} \ell(\tilde{f}^L(x_i;\tilde{\Theta}), y_i). \tag{11.79}$$

A key observation in (11.79) and the new BN model (11.77) is that

$$\begin{aligned} \mu_X^\ell &= \mathbb{E}_{x\sim X}[f^\ell(x)], \\ \sigma_X^\ell &= \mathbb{E}_{x\sim X}\left[(f^\ell(x) - \mathbb{E}_{x\sim X}[f^\ell(x)])^2\right], \\ \mathcal{L}(\tilde{\Theta}) &= \mathbb{E}_{(x,y)\sim(X,Y)}\left[\ell(\tilde{f}^L(x_i;\tilde{\Theta}), y_i)\right]. \end{aligned} \tag{11.80}$$

Here we need to mention that

$$x \sim X$$

means x subject to the discrete distribution of all data X.

11.3.5 BN: some "modified" SGD training algorithm

Following the key observation in (11.80), and recall the similar case in SGD, we do the the sampling trick in (11.79) and obtain the mini-batch SGD:

$$x \sim X \approx x \sim \mathcal{B}, \tag{11.81}$$

here $\mathcal{B}$ is a mini-batch of batch X with $\mathcal{B} \subset X$.

However, for problem in (11.79), it is very difficult to find some subtle sampling method because of the composition of μ_X^ℓ and $[\sigma_X^\ell]$. However, one simple way for sampling (11.79) can be chosen as taking (11.81) for all the expectation case in (11.79) and (11.80).

This is to say, in training process (t-th step for example), once we choose $B_t \subset X$ as the mini-batch, then the model becomes

$$\begin{cases} \tilde{f}^1(x_i) &= W^1 x_i, \\ \tilde{f}^\ell &= W^\ell \sigma_{\mathrm{BN}}(\tilde{f}^{\ell-1}), \quad \ell = 2, ..., L. \end{cases} \tag{11.82}$$

where

$$\sigma_{\mathrm{BN}} := \sigma \circ \mathrm{BN}_{\mathcal{B}_t}, \tag{11.83}$$

or we can say that X is replaced by $\mathcal{B}_t$ in this case.

Here $\mathrm{BN}_{\mathcal{B}_t}$ is defined by

$$\begin{aligned} \mu_{\mathcal{B}_t}^\ell &\leftarrow \frac{1}{m}\sum_{i=1}^{m} f^\ell(x_i) && \text{mini-batch mean} \\ \sigma_{\mathcal{B}_t}^\ell &\leftarrow \frac{1}{m}\sum_{i=1}^{m} (f^\ell(x_i) - \mu_{\mathcal{B}_t})^2 && \text{mini-batch variance} \\ \hat{f}^\ell(x) &\leftarrow \frac{f^\ell(x) - \mu_{\mathcal{B}_t}^\ell}{\sqrt{\sigma_{\mathcal{B}_t}^\ell + \epsilon}} && \text{normalize} \\ \mathrm{BN}_{\mathcal{B}_t}(\tilde{f}^\ell) := \tilde{f}^\ell(x) &\leftarrow \gamma^\ell \hat{f}^\ell(x) + \beta^\ell && \text{scale and shift} \end{aligned} \tag{11.84}$$

Here BN operation introduce some new parameters as γ and β. Thus to say, for training phase, if we choose mini-batch as $\mathcal{B}_t$ in t-th training step, we need to take gradient as

$$\frac{1}{m}\nabla_{\tilde{\theta}} \sum_{i\in\mathcal{B}_t} \ell(\tilde{f}^L(x_i;\tilde{\Theta}), y_i), \tag{11.85}$$

which needs us the to take gradient for μ_B^ℓ or $[\sigma_B^\ell]$ w.r.t w^i for $i \le \ell$.

Questions: To derive the new gradient formula for BN step because of the fact that

$$\mu_{\mathcal{B}_t}^\ell, \quad \text{and} \quad \sigma_{\mathcal{B}_t}^\ell,$$

contain the output of $\tilde{f}^{\ell-1}$.

This is exact the batch normalization method described in [9].

11.3.6 To final model with BN in DNN after training

One key problem is that, in the BN operator, we need to compute the mean and variance in a data set (batch or mini-batch). However, in the inference step, we just input one data into this DNN, how to compute the BN operator in this situation.

Actually, the γ and β parameter is fixed after training, the only problem is to compute the mean μ and variance σ. All the mean $\mu_{\mathcal{B}_t}$ and variance $\sigma_{\mathcal{B}_t}$ during the training phase are just the approximation of the mean and variance of whole batch i.e. μ_X and σ_X as shown in (11.80).

One natural idea might be just use the BN operator w.r.t to the whole training data set, thus to say just compute μ_X and σ_X by definition in (11.73).

However, there are at least the next few problems:

- computation cost,
- ignoring the statistical approximation (don't make use of the $\mu_{\mathcal{B}_t}$ and $\sigma_{\mathcal{B}_t}$ in training phase).

Considering that we have the statistical approximation for μ_X and σ_X during each SGD step, moving average might be a more straightforward way. Thus two say, we define the μ^ℓ and $[\sigma^\ell]$ for the inference (test) phase as

$$\mu^\ell = \frac{1}{T}\sum_{t=1}^{T} \mu_{\mathcal{B}_t}^\ell, \quad \sigma^\ell = \frac{1}{T}\frac{m}{m-1}\sum_{t=1}^{T} \sigma_{\mathcal{B}_t}^\ell. \tag{11.86}$$

Here we take Bessel's correction for unbiased variance. The above moving average step is found in the original paper of BN in [9].

Another way to do this is to call the similar idea in momentum. At each time step we update the running averages for mean and variance using an exponential decay based on the momentum parameter:

$$\begin{aligned} \mu_{\mathcal{B}_t}^\ell &= \alpha\mu_{\mathcal{B}_{t-1}}^\ell + (1-\alpha)\mu_{\mathcal{B}_t}^\ell, \\ \sigma_{\mathcal{B}_t}^\ell &= \alpha\sigma_{\mathcal{B}_{t-1}}^\ell + (1-\alpha)\sigma_{\mathcal{B}_t}^\ell. \end{aligned} \tag{11.87}$$

α is close to 1, we can take it as 0.9 generally. Then we all take bath mean and variance as $\mu_X^\ell \approx \mu_{\mathcal{B}_T}^\ell$ and $\sigma_X^\ell \approx \sigma_{\mathcal{B}_T}^\ell$.

Many people argue that the variance here should also use Bessel's correction.

11.3.7 Batch Normalization for CNN

One key idea in BN is to do normalization with each scalar features (neurons) separately along a mini-batch. Thus to say, we need one to identify what is neuron in CNN. This is a historical problem, some people think neuron in CNN should be the pixel in each channel some thing that each channel is just one neuron. BN choose the later one. **One (most ?) important reason for this choice is the fact of computation cost.**

For convolutional layers, BN additionally wants the normalization to obey the convolutional property – so that different elements of the same feature map, at different locations, are normalized in the same way. To compute $\mu_{\mathcal{B}_t}^\ell$, we take mean of the set of all values in a feature map across both the elements of a mini-batch and spatial locations – so for a mini-batch of size m and feature maps of size $m_\ell \times n_\ell$ (image geometrical size), we use the effective mini-batch of size $mm_\ell n_\ell$. We learn a pair of parameters γ_k and β_k per feature map (k-th channel), rather than per activation.

For simplicity, then have the following BN scheme for CNN

(11.88)

$$
\begin{aligned}
[\mu_{\mathcal{B}_t}^\ell]_j &\leftarrow \frac{1}{m \times m_\ell \times n_\ell} \sum_{i=1}^{m} \sum_{1\le s\le m_\ell, 1\le t\le n_\ell} [f^\ell(x_i)]_{j;st} && \text{mean on channel } j \\
[\sigma_{\mathcal{B}_t}^\ell]_j &\leftarrow \frac{1}{m \times m_\ell \times n_\ell} \sum_{i=1}^{m} \sum_{1\le s\le m_\ell, 1\le t\le n_\ell} ([f^\ell(x_i)]_{j;st} - [\mu_{\mathcal{B}_t}^\ell]_j)^2 && \text{variance on channel } j \\
[\hat{f}^\ell(x)]_{j;st} &\leftarrow \frac{[f^\ell(x)]_{j,st} - [\mu_{\mathcal{B}_t}^\ell]_j}{\sqrt{[\sigma_{\mathcal{B}_t}^\ell]_j + \epsilon}} && \text{normalize} \\
[\mathrm{BN}_{\mathcal{B}_t}(\tilde{f}^\ell)]_{j;st} := [\tilde{f}^\ell(x)]_{j;st} &\leftarrow [\gamma^\ell]_j [\hat{f}^\ell(x)]_{j;st} + [\beta^\ell]_j && \text{scale and shift on channel}
\end{aligned}
$$

11.3.8 Training MgNet with batch normalization

When we train MgNet, we also adopt the BN mechanism. This means that, when we do training for MgNet, we apply the following basic block in MgNet

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + \mathrm{ReLU} \circ \mathrm{BN}_{\mathcal{B}_t} \circ B^{\ell,i} * \mathrm{ReLU} \circ \mathrm{BN}_{\mathcal{B}_t}(f^\ell - A^\ell * u^{\ell,i-1}), \tag{11.89}$$

or simply

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + \mathrm{ReLU}_{\mathrm{BN}} \circ B^{\ell,i} * \mathrm{ReLU}_{\mathrm{BN}}(f^\ell - A^\ell * u^{\ell,i-1}), \tag{11.90}$$

where

$$\mathrm{ReLU}_{\mathrm{BN}} = \mathrm{ReLU} \circ \mathrm{BN}_{\mathcal{B}_t}. \tag{11.91}$$

References

[1] W. L. Briggs, V. E. Henson, and S. F. Mccormick. *A multigrid tutorial: second edition*. Society for Industrial and Applied Mathematics, 2000.

[2] L. Chen, P. Sun, and J. Xu. Optimal anisotropic meshes for minimizing interpolation errors in l^p-norm. *Mathematics of Computation*, 76(257):179–204, 2007.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[5] J. He, Y. Chen, and J. Xu. Constrained linear data-feature mapping for image classification. *arXiv preprint arXiv:1911.10428*, 2019.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, et al. Neural networks: Tricks of the trade. *Springer Lecture Notes in Computer Sciences*, 1524(5-50):6, 1998.

[14] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[15] S. G. Nash. A multigrid approach to discretized optimization problems. *Optimization Methods and Software*, 14(1-2):99–116, 2000.

[16] A. Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

[17] L. R. Scott and S. Zhang. Finite element interpolation of nonsmooth functions satisfying boundary conditions. *Mathematics of Computation*, 54(190):483–493, 1990.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] M. H. Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(5):237–254, 1948.

[20] R. S. Strichartz. *A guide to distribution theory and Fourier transforms*. World Scientific Publishing Company, 2003.

[21] X.-C. Tai and J. Xu. Global and uniform convergence of subspace correction methods for some convex optimization problems. *Mathematics of Computation*, 71(237):105–124, 2002.

[22] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Academic press, 2000.

[23] S. Wiesler and H. Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pages 657–665, 2011.

[24] J. Xu. Finite element methods. Lecture notes, 2020.

[25] J. Xu and L. Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017.