工学系の卒論生のための数式記述入門

東京大学　情報理工学系研究科　講師　松井勇佑

2021 年 1 月 7 日

- 本資料の GitHub リポジトリ： https://github.com/matsui528/math_writing
- 著者のウェブページ：http://yusukematsui.me

1　はじめに

本資料は、初めて技術文章を書く学生向けの、数式記述の入門書です。自分のアイデアを式として記述する作業は難しく、最初のうちはうまく書けません。本資料は、そんな学生向けに、数式をどう記述すればいいかのガイドラインを示します。特に、よくありがちな、**変数がゴチャゴチャしてわかりにくく**なったり、**アルファベットを使いすぎて足りなくなったり**する状況の改善を目指します。

重要なことは次の三点です。

- 伝えたいことを間違えることなく正確に記述すること
- 複雑すぎず、しかし曖昧でもない、必要十分な記述とすること
- 10 年後に読まれても理解できる記述にすること

まず、式を間違えると読者が混乱します。そして、間違えた式に基づいた議論は間違いです。なので、正確な記述をしましょう。次に、式が複雑すぎたり曖昧だと読者が混乱します。複雑すぎる式は訓練で簡潔に出来ます。本記事はそのための指南書です。最後に、未来の読者でも理解できる記述にしましょう。なぜなら、研究とは知識を積み重ねる「時代を超えた共同作業」だからです。numpy のブロードキャストを前提とする記法のような、現在使われているライブラリを知らないと理解できない記述はやめましょう。

本資料は読者として工学系の卒論生を想定しています。特に、筆者の専門である情報工学、そのなかでもコンピュータビジョン・画像処理分野の学生を主たる対象としています[*1]。**分野が違うと慣習も大きく違う**ので注意してください。本資料の内容と普段読んでいる論文の記述に矛盾があった場合は、常に自分の分野の慣習を優先してください。また、コメントなど歓迎です。issue へ書き込むか、pull request をお願いします。

1.1　参考資料

まず、数式の記述を考える前に、アカデミックライティングの基本を前提知識として勉強しておきましょう。以下が必読でので読んでおきましょう。

- 木下是雄、理科系の作文技術：アカデミックライティングに関する古典的な名著です。薄い新書なので、とりあえず読みましょう。
- 東京大学松尾先生による記事：読み物としても面白い、松尾先生による論文執筆指南です。短い文章で重要な事項が凝縮されており、必読です。
  - 松尾ぐみの論文の書き方
  - 松尾ぐみの論文の書き方：英語論文

次に、数式の表記（notation）に関して参考になる資料を紹介します。本記事では、これらを参考に、松井の考える notation のベストプラクティスを紹介します。

- D. A. ハーヴィル、統計のための行列代数　上下：これは線形代数の辞書です。表記はもとより、導出がわからない式に出会った場合に参考になります。
- Golub ら、Matrix Computations：これは実際に計算することを念頭においた、線形代数の教科書です。こちらも、辞書として使えます。
- Peterson ら、The Matrix Cookbook：これは線形代数の公式のクックブックです。無料の pdf として公開されており、簡単に調べたいときはこれが参考になります。
- Goodfellow らの Deep Learning の notation ページ：Deep Learning に関する教科書の notation の部分が無料で公開されています。画像処理分野の表記方法として参考になります。
- 名工大横田先生による「テンソル分解の基礎と画像・信号処理への応用」：行列を一般化した「テンソル」に関する、日本語での包括的な資料です。

また、TeX に関しては、ネット上の資料が古かったり、現代的なベストプラクティスがわからないことが多々あります。Overleaf の TeX 入門ページが辞書として参考になります。

1.2　アルファベットが足りなくなるとは

本記事ではよく「アルファベットが足りなくなる」とか、「アルファベットを消費しないために」といった表現をします。これは、論文中で変数や定数にアルファベットを使っていくうちに、自然に使えるアルファベットを使い切ってしまう状況を意味します。「たいしたことを言っていないのに何故かアルファベットが全然足りない」という状況は多くの人が一度は経験すると思います。本記事では、それを防ぐ様々な方法を紹介します。

2　変数と定数の表記

変数・定数の表記をしっかり行うことは極めて重要です。表記がしっかりしていないと、読者は混乱します。逆に、表記がしっかりしていれば、それだけで読みやすく意図が伝わりやすい記述になります。ブレない表記を目指しましょう。

変数や定数の表記は分野によって慣習が大きく違います。まず

[*1] ディープラーニング時代の画像処理分野の論文中には何も考えずに numpy 表記をコピーしただけの間違った数式があまりにも多いです。私はそういう論文を査読する際「もうちょっとしっかりしてくれよ」と常々思っていました。これからの学生にはそのような間違いを犯してほしくないので、この文章を書くことにしました

は自分の分野の慣習を理解し、それに従ってください。重要なのは**統一する**ことです。例えば本文中でベクトルは太字で書くと決めたら、そのルールをずっと守ってください。表記のルールが統一されていない文章は、不明瞭で誤解を招きます。

2.1 まずはじめに：実数や自然数

まずはじめに、黒板太字（\mathbb）を考えましょう。黒板文字は、いくつかの慣習的な表記で用います。実数全体を表す $\mathbb{R}$、整数全体を表す $\mathbb{Z}$、自然数全体を表す $\mathbb{N}$ などです。他の例は wikipedia 等を参照してください。ときどき黒板太字を普通のベクトルに使う論文を見ますが、黒板太字は上記のような慣用表記のみに使うことをオススメします。なぜなら、上記のような慣用表記に黒板太字を使うことが慣習である以上、その表記をベクトルという別概念に使うことは混乱を招くからです。

ちなみに、パワーポイントで黒板太字を用いるには、数式機能で\doubleRとします。これは TeX における\mathbb{R}と同じ表記になります。

さて、黒板文字を理解したうえで、画像処理分野でのオススメ表記ルールを表 1 にまとめました。これらについて一つ一つ見ていきましょう。

2.2 スカラー

まずはじめにスカラー値です。これ何も装飾せずに、小文字あるいは大文字を使うことをオススメします。ここで、ドメインを明記するとわかりやすくなります。ドメインとは、その値が定義される範囲です。たとえば「ここで、重み $w \in \mathbb{R}$ を考えます」というような形です。これにより、w は実数であることが明示できます。いくつかの離散値の中からどれか一つを値として取る場合は、集合の要素を示す「$\in$」、および集合そのものを表記します。例えば次のような形です。「ここで、係数 α は三通りの値のいずれかを取ります。：$\alpha \in \{0.1, 0.5, 1.0\}$」

ここで、区間を示すカッコの使い方を復習しておきましょう。これらは大学入試で習っていると思うのですが、いざ自分で数式を書こうと思うとなかなか正確に書きづらいようです。

- $a \in [2, 7]$: a は 2 以上 7 以下の実数
- $a \in (2, 7)$: a は 2 より大きく 7 より小さい実数
- $a \in [2, 7)$: a は 2 以上で 7 より小さい実数
- $a \in \{2, 7\}$: a は 2 か 7
- $a \in \{2, \ldots, 7\}$: 自然に解釈すれば a は 2 か 3 か 4 か 5 か 6 か 7。しかし文脈から別の意味にとられることもある。整数列挙であることを強調するなら $\{2, 3, \ldots, 7\}$ としてもいい。

最後の例は必ずしも整数列挙だけではない点に注意しましょう。例えば「2 以上 10 以下の 2 の倍数の集合 $\mathcal{N} = \{2, \ldots, 10\}$ を考える」と明示的に述べた場合、$\mathcal{N}$ は $\{2, 4, 6, 8, 10\}$ と理解するほうが自然であり、$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ とは取られないと思います。

2.3 ベクトル

次にベクトルです。これは、**小文字で太字**（\mathbfか\bm）にすることをオススメします。ベクトルを太字にしない流派はいくらでもあります。しかし、太字にすると心に決めて全て統一するほうが読みやすいというのが私の考えです。「これはいちいちベクトルだと主張しないでも自明だろう」と著者で思ったとしても、読者側からすると全く自明ではないケースがいくらでもあるように思います。後述するように行列には大文字を使うことをオススメしますので、ベクトルは小文字で統一するほうがわかりやすいと思います。

重要なことは、意図的に隠したい場合を除き、**常にドメインを明示すること**です。すなわち、

- D 次元の入力ベクトル $\mathbf{x}$ を考える

ではなく、

- D 次元の入力ベクトル $\mathbf{x} \in \mathbb{R}^D$ を考える

としましょう。ベクトルがいったい何なのかということは、著者が思っているよりもずっと、読者に伝わりづらいです。ドメインを明示することでずっとわかりやすくなります。また、上記からわかるように、実質的な記述量の増加は「$\in \mathbb{R}^D$」だけなので、記述が増えて困るということもありません。特殊な理由がある場合を除き、全てのベクトルに関して、初めて登場するときに必ずドメインを明示することをオススメします。

\mathbfと\bmのどちらを使うかは、ケースバイケースです。

- 太字なし: x, a, 1, μ
- \mathbfの例: $\mathbf{x}$, $\mathbf{a}$, $\mathbf{1}$, μ
- \bmの例: $\boldsymbol{x}$, $\boldsymbol{a}$, $\mathbf{1}$, $\boldsymbol{\mu}$

上からわかるように、\mathbfのほうが、標準文字と違った書体（非イタリック）になります。\bmのほうは、標準文字をそのまま太くしたような書体（イタリックのまま）です。ここで使われる具体的なフォントは論文のフォーマットによって変わるので注意してください。μ のような特殊文字は\mathbfで太字にならない（非イタリックにならない）ので、そういう特殊文字のベクトルを多用する場合は\bmでそろえるほうが綺麗だと思います。一方で、\bmは太字に見えづらい面もあります。すわなち、pdf ビューアの種類、ディスプレイのサイズ、紙面への印刷具合などで、太字ではなく通常文字に見えることがあります。なので、太字であることを強調するためにあえて\mathbfにするパターンもあります。実際に執筆する論文のフォーマット上で両方を表示して比較して決めるとよいと思います。私は最近は CVPR フォーマットでは\mathbfを使うようにしています。ちなみに、パワーポイントで太字を表現したいときは、文字を選択して太字にすることで表現できます。

また、ベクトルは、断りが無い限り「縦（列）ベクトル」である、と心に決めるとよいです。その場合、数字を横にならべて値を表現する場合は転置する必要があることに注意してください。

- $\mathbf{x} = [1, 2, 3]^\top$ ：縦ベクトル。OK
- $\mathbf{x} = [1, 2, 3]$ ： 横ベクトル。明示的に横ベクトルを作るとき以外、この表記は使わない

転置がないと「横（行）ベクトル」になってしまいますね。ちなみに転置記号としては「T」を使う人が多いですが、上式のように、TeX で用意されている\topを使うと綺麗です。このように縦か横かにこだわる理由は、行列演算を行う際に注意が必要だからです。すなわち、$\mathbf{x} \in \mathbb{R}^2$ が縦ベクトルであるときは行列 $\mathbf{A} \in \mathbb{R}^{2\times 2}$ に対し $\mathbf{Ax}$ が計算できますが、$\mathbf{x} \in \mathbb{R}^2$ が横ベクトルであるときは計算できません。このあたりを厳密にするために、「特に断らなければベクトルは縦ベクトル」と心に決めて記述すると良いです。

より厳密に言うと、$\mathbb{R}^D$ の表記だけでは横か縦かは定義されてい

型	表記	ドメインの例	値の例
スカラー	小文字か大文字。何もつけない。	$a \in \mathbb{R}$. $b \in \mathbb{N}$. $K \in \{10, 20, 30\}$.	$a = 3.2$. $b = 13$. $K = 20$.
ベクトル	小文字で太字（\mathbfか\bm）	$\mathbf{x} \in \mathbb{R}^3$. $\boldsymbol{x} \in \mathbb{R}^D$. $\mathbf{b} \in \{0, 1\}^B$.	$\mathbf{x} = [0.1, 0.2, 0.3]^\top$. $\mathbf{b} = [0, 1, 1, 0, 1]^\top$.
行列 （および三階以上のテンソル）	大文字で太字（\mathbfか\bm）	$\mathbf{A} \in \mathbb{R}^{2\times3}$. $\mathbf{I} \in [0, 1]^{H\times W\times3}$.	$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$.
集合	大文字でカリグラフィ（\mathcal）	$\mathcal{S} \subset \mathbb{N}$.	$\mathcal{S} = \{2, 4, 8, 16\}$.

表 1　変数・定数の表記

ません。なので、$\mathbb{R}^D$ を、列ベクトルであることを明示する $\mathbb{R}^{D\times1}$ の省略表記であると考えるとスッキリすると思います。

- $\mathbf{a} \in \mathbb{R}^D$: この表記の場合、縦ベクトルだと心に決める。これは $\mathbb{R}^{D\times1}$ の省略版だとも解釈できる
- $\mathbf{b} \in \mathbb{R}^{D\times1}$: 縦ベクトルであると明示する場合
- $\mathbf{c} \in \mathbb{R}^{1\times D}$: 横ベクトルであると明示する場合

このあたりは画像処理分野の慣習かもしれません。いずれにせよ、**表記の矛盾を無くし、表記を統一する**ことが重要です。

2.4　行列、および三階以上のテンソル

次に行列、および三階以上のテンソルです。三階以上のテンソルというのは、$\mathbf{A} \in \mathbb{R}^{5\times5\times5}$ のように行列を一般化したものです。このような表記は画像分野では多用します。なぜなら、そもそも画像そのものを正確に記述しようとするとそのような表記になるからです。すなわち、縦 H 横 W の 3 チャンネル（RGB）画像 $\mathbf{I}$ は $\mathbf{I} \in \mathbb{R}^{H\times W\times3}$ のように書けます[*2]。また、近年のディープラーニングに関する議論では、処理中に出てくる重み情報を、上記のように階数が大きいテンソルとして表現することが多いです。テンソルに関しては名工大横田先生の資料[*3]をご参照ください。

さて、これらは、基本的に**大文字で太字**にすることをオススメします。行列は太字にしないほうが一般的かもしれません。しかし、私は太字にすることをオススメします。というのも、太字にしない場合はスカラーと混合される可能性が残るからです。なので、情報が複数含まれるもの（ベクトルや行列）は太字にする、と決めるほうがスッキリすると思います。

ちなみに、例えば Goodfellow らの「Deep Learning」では三階以上のテンソルに\mathsfを割り当てています[*4]。もし、三階以上のテンソルを多用したり、行列と区別をもたせることが重要な場合は、そのように別の書式を割り振るほうが良い場面もあると思います。

さて、それではいくつか例を見てみましょう。

- $a \in \mathbb{R}$: 実数のスカラー
- $\mathbf{a} \in \mathbb{R}^2$: 実数 2 つを並べたベクトル
- $\mathbf{A} \in \mathbb{R}^{2\times3}$: 実数が 2×3 個並んだ行列
- $\mathbf{B} \in [0, 1]^{2\times3}$: 要素が全て 0 以上 1 以下である、$2 \times 3$ の行列
- $\mathbf{C} \in [0, 1)^{2\times3\times4}$: 要素が全て 0 以上 1 未満である、$2 \times 3 \times 4$ のテンソル
- $\mathbf{D} \in \{-1, 0, 1\}^{2\times3}$: 要素が全て「-1, 0, 1」の何れかである、2×3 の行列
- $\mathbf{E} \in \{0, \ldots, 100\}^{2\times3}$: 要素が全て「$0, 1, \ldots, 100$」の何れかである、$2 \times 3$ の行列
- $\mathbf{f} \in \mathbb{N}^{1\times ab}$: 要素が自然数で、幅が ab の横ベクトル
- $\mathbf{F} \in \mathbb{N}^{a\times b}$: 要素が自然数で、$a \times b$ の行列

2.5　集合

最後に集合です。集合は大文字でカリグラフィ（\mathcal）にするといいでしょう。例えば D 次元のベクトル N 本の集合であるデータセット $\mathcal{X}$ を考えると、これは $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$、あるいは $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$ のように書けます。ここで $\mathcal{X} \subset \mathbb{R}^D$ であり、$\mathbf{x}_n \in \mathbb{R}^D$ です。「集合の部分集合」を示すには「$\subset$」あるいは「$\subseteq$」を使い、「集合の要素」を表すには「$\in$」を使う、という点を混乱しないようにしましょう。ここでは、$\mathbf{x}_n \in \mathcal{X}$ でもあります。また、集合の要素数を表すには $|\mathcal{X}|$ とします。ここでは $|\mathcal{X}| = N$ ですね。

ちなみに、集合というと普通は重複を含みません。重複を含むものは多重集合：multiset と言います。なので、上記のようにデータセットを定義すると、データセット中に重複するベクトルがあるときに扱いに困る（$|\mathcal{X}| \neq N$ になる）はずなのですが、画像処理の分野ではこのあたりが厳密に扱われないようです。

パワーポイント中で集合を表現するには、数式機能中で\scriptSのようにします。これは TeX 中の\mathrm{S}に対応します。

2.6　例

上記のルールを厳密に守ると、人工的な例ですが例えば以下のように「c」というアルファベット一字で様々なものを表現することができます。

k-means クラスタリングにより作られた C 本の D 次元ベクトル（中心ベクトル）を、次のように表記する。

$$\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_C\} \subset \mathbb{R}^D. \tag{1}$$

ここで、$\mathbf{c}_i \in \mathbb{R}^D$ は i 番目の中心ベクトルをを指し、$\mathcal{C}$ は C 本の中心ベクトルを集めた集合である。また、各ベクトルを横に並べて作った $D \times C$ の行列を次のように定義する。

$$\mathbf{C} = [\mathbf{c}_1 \mid \mathbf{c}_2 \mid \cdots \mid \mathbf{c}_C] \in \mathbb{R}^{D\times C}. \tag{2}$$

上の例では C と $\mathcal{C}$ が区別しづらいのでちょっとやりすぎですが、C を例えば K にすれば十分に見やすいです。上のように\mathcalでベクトルの集合を作り、それを並べた行列は同じアルファベットを\mathbfしたものにする、という表記は、使うアル

[*2] ちなみにここでは $\mathbb{R}^{3\times H\times W}$ のようにチャンネル情報を前にしてももちろん OK です。あるいは、H と W を入れ替えてもいいです。
[*3] https://www.slideshare.net/yokotatsuya/2014-3-13
[*4] https://www.deeplearningbook.org/contents/notation.html

ファベットを減らしつつ関係性を示せる小ネタです。

3　関数の入出力の表記

関数を定義するとき、その入力と出力も明示するとわかりやすい場面が多々あります。このとき、(1) 入力の集合と出力の集合を明記、(2) 入力の要素と出力の要素を明記、とする二通りの形式があります。場面に応じて使い分けると良いでしょう。

3.1　関数の入出力の表記の例

例えば、$x \in \mathbb{R}$ に関する関数 $f(x) = x^2$ を考えます。この入出力を集合を用いて表記すると次のようになります。

$$f : \mathbb{R} \to \mathbb{R}. \tag{3}$$

これを要素で表記すると次のようになります。

$$f : x \mapsto x^2. \tag{4}$$

要素で表記するときは`\mapsto`を使う点に注意してください。

多変数の例を見てみましょう。また、出力変数も考えてみます。ここで、x と y に関する関数 $f(x, y) = z = x^2 + y + 1$ は次のように書けます。集合で表記：

$$f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}. \tag{5}$$

要素で表記：

$$f : (x, y) \mapsto z. \tag{6}$$

$f : (x, y) \mapsto x^2 + y + 1$ のように全て書くほうがスッキリする場合もあると思います。

ベクトルの例も見てみましょう。定数ベクトル $\mathbf{a} \in \mathbb{R}^D$ および定数のスカラー値 $b \in \mathbb{R}$ があるとします。ここで変数 $\mathbf{x} \in \mathbb{R}^D$ に関する関数 $f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} + b$ は次のように書けます。集合で表記：

$$f : \mathbb{R}^D \to \mathbb{R}. \tag{7}$$

要素で表記：

$$f : \mathbf{x} \mapsto \mathbf{a}^\top \mathbf{x} + b. \tag{8}$$

入力も出力もベクトルの例も見てみましょう。$\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^2$ とします。ここで 3 要素を返す関数 $f(\mathbf{x})$ を要素表記で見てみましょう。

$$f : \mathbf{x} \mapsto \begin{bmatrix} x_1 + x_2 \\ 3x_1 + \log x_2 \\ x_2^3 \end{bmatrix} \tag{9}$$

このように、要素表記で書き下すスタイルは、定義そのものとも言えますね。これを入出力の集合で書くと次のようになります。

$$f : \mathbb{R}^2 \to \mathbb{R}^3 \tag{10}$$

3.2　関数は太字にすべきか

出力がベクトルのときに関数を太字にするかどうかは、ケースバイケースだと思います。すなわち、

$$\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^3 \tag{11}$$

とする例です。こうすると $\mathbf{z} = \mathbf{a} + \mathbf{f}(\mathbf{x})$ のようにベクトルに関する数式がすべて太字になり、美しい場面もあります。一方で、近年のコンピュータビジョンの論文中の式では「論文中に出現する全ての関数はベクトルを出力する」といった場面も多々あります。そのような場合、すべてが太字になっているとちょっとうるさい、と感じるかもしれません。よって、ベクトル出力の関数を太字にするかどうかはケースバイケースだと思います。

3.3　要素形式か集合形式か

集合と要素はどちらの形式で書く方がよいのでしょうか？ 多くの場合は集合で表記するほうが有用だと思います。なぜなら、読者が知りたいのは入力と出力として取りうる値が何なのか、という点だからです。集合で書けば、入力と出力が取り得る範囲を明示的に述べることができます。要素の表記は、そのインスタンスを述べているに過ぎません。

一方で、要素の表記のほうがわかりやすい、「物体検出器」の例も見てみましょう。以下はちょっとうるさい人工的な例であり実際はこのように書く人は少ないと思いますが、例として紹介します。各画素が 0 から 255 までの値をとり、RGB の 3 チャンネルをもつ画像 $\mathbf{I} \in \{0, \ldots, 255\}^{H \times W \times 3}$ を考えます。ここで H, W は画像の高さと幅です。この画像を物体検出器 f に入力することを考えます。出力は K クラス認識結果を表すラベル $l \in \{1, \ldots, K\}$、検出領域のバウンディングボックス $\mathbf{b} = [y, x, h, w]^\top \in \mathbb{N}^4$、およびその確信度 $\alpha \in \mathbb{R}$ とします。ここでバウンディングボックスとは画像中の注目領域を矩形として指定したものです。矩形を表現するために、4 つのスカラー値が用いられます。ここで y, x はバウンディングボックスの左上の座標値、h, w はバウンディングボックスの幅と高さだとします。これを集合で正確に表記しようとすると

$$f : \{0, \ldots, 255\}^{H \times W \times 3} \to \{1, \ldots, K\} \times \mathbb{N}^4 \times \mathbb{R} \tag{12}$$

となります。この記述は厳密ですが、次のような問題もあります。まず、H や W といった値は本質的に重要な概念では無いかもしれません。であれば、そのためだけにわざわざアルファベットを 2 つ消費したうえにそれらについて説明の文章を書くメリットが無いかもしれません。また、画素値を述べることの意味も薄いかもしれません。なぜなら、画素の範囲を 0 から 255 に設定したのは単に実装上の都合であり、「0 から 1」と定義しても一般的には構わないからです。ここでは、要素表記を採用するとそのような部分を抽象化できます。例えば次のようにします。

> K クラス識別の物体検出器 $f : \mathbf{I} \mapsto (l, \mathbf{b}, \alpha)$ を考える。これは画像 $\mathbf{I}$ を受け取り、ラベル $l \in \{1, \ldots, K\}$、バウンディングボックス $\mathbf{b} = [y, x, h, w]^\top \in \mathbb{N}^4$、およびその確信度 $\alpha \in \mathbb{R}$ を返す。

これらは、「意図的に変数のドメインを隠したい例」だと言えます。

4　集合の表記

N 個の要素の集合を表す表記 $\{x_1, x_2, \ldots, x_N\}$ を省略して記述したいとき、つい以下のように書きがちです。

$$\{x_i \mid i = 1, 2, \ldots, N\} \tag{13}$$

あるいは、

$$\{x_i \mid 1 \leq i \leq N\} \tag{14}$$

あるいは、

$$\{x_i \mid i \in \{1, 2, \ldots, N\}\} \tag{15}$$

などです。これらでも構わないのですが、この場合は良く知られている省略記法を用いて次のように書くとスッキリするでしょう。

$$\{x_i\}_{i=1}^N \tag{16}$$

ここでは、下付き文字でインデクスの始まりを定義し、上付き文字でその終わりを示します。この表記は、言われてみれば当たり前なのですが、案外思いつかない場面が多いようです。

ここで知っておくと便利かもしれない小ネタを紹介します。集合を表記する際、その大きさは重要でなかったり決定できない場合があります。あるいはアルファベットが足りないので「N」というようにアルファベットを割り振りたくないかもしれません。その場合は、「宣言時には最終要素を明示しない。要素数が必要なときは $|\cdot|$ の表記を使う」という方式が便利です。すなわち、集合 $\mathcal{X}$ を次のように定義します。

$$\mathcal{X} = \{x_1, x_2, \dots\} \tag{17}$$

そして、この要素数は $|\mathcal{X}|$ のように書けます。

この方式を使うと、集合そのものにアルファベットが必要になる一方で、集合の個数を示すアルファベットは必要なくなります。これは例えば次のように複数の集合を考える場合に便利です。3 つの集合を考えます。それをそれぞれ

$$\{x_i\}_{i=1}^{N}, \quad \{y_i\}_{i=1}^{M}, \quad \{z_i\}_{i=1}^{K}. \tag{18}$$

のように定義してしまうと、 N, M, K といった 3 つの関係ないアルファベットを消費してしまいます。また、インデックスを表す i が共有されてしまい、不都合が生じるかもしれません。かといってインデックスを i, j, k のように使ってしまうと、それもまたアルファベットを消費します。これを次のように定義すれば、アルファベットは要素との対応をもった mathcal で表現でき、直感的です。

$$\mathcal{X} = \{x_1, x_2, \dots\}, \quad \mathcal{Y} = \{y_1, y_2, \dots\}, \quad \mathcal{Z} = \{z_1, z_2, \dots\}. \tag{19}$$

そして、要素数が必要になったときは $|\mathcal{X}|$ を使えます。

5　下付き文字・上付き文字

可読性を下げるので、**変数に対する下付き文字・上付き文字は可能な限り減らす**ことをおすすめします。例えば

$$x_{i,j}^{k} \tag{20}$$

のような変数は、i, j, k が何かを考えながら読む必要があります。出来るだけそれらを消せないか考えるとよいです。また、

$$x_{a_i} \tag{21}$$

のように「下付きの下付き」は可能な限りやめたほうが良いです。これは x_a 対し i が下付き文字なのか、x に対し a_i が下付き文字なのか区別がつきません。

下付き文字を減らせる一つの例はループ周回のインデクスです。例えば次の Python コードを考えましょう。

```python
V = [1.0, 2.0, 4.0, 8.0]

for n in len(V):
    print(V[n])   # (1)

for v in V:
    print(v)      # (2)
```

ここで、(1) はループインデクスを用意して要素にアクセスしています。一方で、(2) は集合の要素を直接イテレーションする形式です。Python では、無駄な変数 i を使わない (2) のほうが推奨されますね。これと同じことが数式の表記でも言えます。例えばデータ $\mathcal{V} = \{v_n\}_{n=1}^{N}$ を考えます。ここで $\mathcal{V}$ の要素に対し、インデクス n に依存しない処理を述べたいとします。このとき、

- v_n に対し関数 f を適用した $f(v_n)$ を・・

のようにするのではなく、

- ここで $\mathcal{V}$ 中の要素 $v \in \mathcal{V}$ について考える。これに関数 f を適用した $f(v)$ を・・

のように言い換えれば、ループインデクスを減らせます。

また、うっかり下付き文字を太字にしないように注意しましょう。これは相当頻繁におきるミスです。

- `\mathbf{x}_i`: $\mathbf{x}_i$: OK
- `\mathbf{x_i}`: $\mathbf{x_i}$: ダメ

下付き文字を太字にすると、「太字すなわちベクトルによって要素指定がされる」と解釈されてしまいます。たとえば、もし別の部分で $\mathbf{i} = [1, 2]^\top$ のような変数が定義されていれば、下付き文字にそれを使ったのか？ すなわち、$\mathbf{x}_{1,2}$ と言いたいのか？ と思われてしまいます。

また、二文字以上のアルファベットをラベルの意味で下付きないし上付きにつけるときは、ローマン（`\mathrm`）にすることをオススメします。

- `x_\mathrm{in}`: x_{in}: OK
- `x_{in}`: x_{in}: 推奨しない

これは、まず、ローマンにしない場合は見にくいです。そして、ローマンにしない場合は、上記の場合 i と n という変数があってその積の値 $i \cdot n$ が下付き文字にきているのか、と解釈されてしまいます。そのような誤解を招かないために、ラベルのアルファベットはローマンにすることをすすめます。

6　英単語をそのまま数式中で使わない

慣習となっている場合を除き、関数名に英単語をそのまま使わないほうがいいです。

- $y = s(x) + 10$: OK
- $y = score(x) + 10$: ダメ

これは下付き文字の場合と同じ理由です。数式環境はアルファベットを並べて英単語を表現することを想定していないため、上でわかるように、イタリックが並んだ表記は見にくいです。そして、上の例だと、$score$ は $s \cdot c \cdot o \cdot r \cdot e$ だと解釈出来てしまいます。なので、英単語をそのまま使うことはやめて、事前に関数として定義しましょう。どうしても英単語などを使いたいときは、せめてローマンにして、つながっている単語だとアピールすることをオススメします。たとえば、バッチノーマライゼーションを表す関数を考えるとき、$BN(x)$ よりも、$\mathrm{BN}(x)$ のほうがまだ良いと思います。なぜなら、$BN(x)$ は $B \cdot N(x)$ だと読めてしまうので。

7　numpy 表記に引っ張られない

numpy はブロードキャストという機能により、「ベクトルとスカラー」のようにドメインが違う変数同士でも演算が可能です。しかし、それは数式では自明ではありません。よって、そのような**numpy 固有の表記を使わないように注意しましょう。**

7.1　違ったドメイン同士の演算をしない

査読でよく見る例を紹介します。ベクトル b（太字にしていない！）に対し、閾値処理をしようと思い、b の全ての要素からスカラー τ を引いたものを考えます。これをうっかり次のように書く人が多いです。

$$b - \tau \tag{22}$$

これは、numpy では可能です。すなわち、τ がブロードキャストされ、b 中の全ての要素に対する演算となってくれます。しかし、そのような表記は数式では不可能です。値を代入してみるとわかります。$b = [10, 20, 30]^\top$, $\tau = 5$ とすると、

$$\begin{bmatrix}10\\20\\30\end{bmatrix} - 5 \tag{23}$$

となります。このようにベクトルからスカラーを引くことはおかしいですよね。しかし、なぜか numpy 表記に引っ張られて上のように書いてしまう人が多いのです。上を正しく記述するには、b をちゃんと太字にしたうえで、要素が全て 1 であるベクトル $\mathbf{1} = [1, 1, \ldots, 1]^\top$ を導入し、

$$\mathbf{b} - \tau\mathbf{1} \tag{24}$$

とすると正しい式になります。このようなミスを防ぐためにも、ベクトルは太字にすることを推奨します[*5]。

7.2　具体的な失敗例

この発展版として、極めてよく見る次のようなケースを考えましょう。以下、特徴マップは高さ 3, 幅 4, チャンネル数 2 だとします。

> ある CNN の出力の特徴マップを X、別の CNN の出力特徴マップを Y とする。ここで要素が 0 か 1 のマスク行列 B を考える。マスクの値が 1 のときは X、0 のときは Y を採用することにより、X と Y を結合した結果のマップである Z を次のように計算する。
>
> $$Z = BX + (1 - B)Y \tag{25}$$

というような形です。これは、numpy の表記を写しただけの、**非常に良くない表現**になっています。まず、なぜ人間は上のような式を書いてしまうのか説明します。上の式は、以下のような numpy の気持ちの表明になっています。

```
In [1]: import numpy as np
In [2]: X = np.arange(24).reshape(2, 3, 4)
In [3]: X
Out[3]:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])

In [4]: Y = -np.arange(24).reshape(2, 3, 4)
In [5]: Y
Out[5]:
array([[[  0,  -1,  -2,  -3],
        [ -4,  -5,  -6,  -7],
        [ -8,  -9, -10, -11]],

       [[-12, -13, -14, -15],
        [-16, -17, -18, -19],
        [-20, -21, -22, -23]]])

In [6]: B = np.array([1, 1, 1, 0, 0, 1, 0,
 ↪  0, 0, 0, 0, 0]).reshape(3, 4)
In [7]: B
Out[7]:
array([[1, 1, 1, 0],
       [0, 1, 0, 0],
       [0, 0, 0, 0]])

In [8]: Z = B * X + (1 - B) * Y
In [9]: Z
Out[9]:
array([[[  0,   1,   2,  -3],
        [ -4,   5,  -6,  -7],
        [ -8,  -9, -10, -11]],

       [[ 12,  13,  14, -15],
        [-16,  17, -18, -19],
        [-20, -21, -22, -23]]])
```

ここでは、X, Y, Z は高さ 3、幅 4、チャンネル数 2 の特徴マップを仮定しています。マスク B は高さ 3、幅 4 の行列です。「T」字型が左上にあるようなマスクになっています。このマスクを X に適用することで、各チャンネルについて、「T」の部分を抜き出します。また、マスクが適用されない位置は Y の値を持ってきます。これらを足し合わせた Z はまさにそのようになっています。なので、31 行目の $Z = BX + (1 - B)Y$ を数式として記述して終わり。という思考になっています。

ここでは**3 点の間違い**があります。これを順番に説明します。まず、ドメインを書けるのであれば書いたほうがいいです。そして、太字にしましょう。ここでは $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbb{R}^{2\times3\times4}$ であり、$\mathbf{B} \in \{0, 1\}^{3\times4}$ が言いたいことでしょう。ここで、$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ はチャンネルの軸である「2」を持っていますが、$\mathbf{B}$ は持っていません。そして、3 点の間違いは以下です。

[*5] ちなみに、四元数の計算などではスカラーとベクトルを足すような表記をとることもあります。

- まず、「$(1-B)$」の差の表記は間違いです。これは**スカラから行列を引いてしまっています**。上で学んだ通り、要素が全て 1 の行列 $\mathbf{1}$ を定義して、$(\mathbf{1}-\mathbf{B})$ としましょう。このようなミスを減らすためにも、やはりベクトルや行列は太字にすることをすすめます。
- 次に、「BX」の積の表記は間違いです。特に断りがなければ、**行列同士の積は行列積を意味する**ことを思い出しましょう。ここで述べたい演算は**要素同士の積**ですね。なので、ここでは要素同士の積（アダマール積）の記号「$\odot$」を定義したうえで、$\mathbf{B}\odot\mathbf{X}$ とする必要があります。
- ですが、実はこれはまだチャンネルを考慮していないので、間違いです。すなわち、$\mathbf{B}$ と $\mathbf{X}$ がともに同じドメインの要素（例えば $\mathbf{B},\mathbf{X}\in\mathbb{R}^{3\times4}$）であればアダマール積が計算できます。しかし今回は $\mathbf{X}$ は複数チャンネルを持っています（$\mathbf{X}\in\mathbb{R}^{2\times3\times4}$）。どうすればよいでしょうか。

上記 3 点の問題を解決する表記は例えば以下になります。いずれも、アダマール積、および全てが 1 の行列は既に定義されていると仮定します。一つ目の方法は、各チャンネルに注目する方式です：

> ある CNN の出力の特徴マップのうち c チャンネル目を $\mathbf{X}\in\mathbb{R}^{3\times4}$ とし、同様に別の CNN の特徴マップの c チャンネル目を $\mathbf{Y}\in\mathbb{R}^{3\times4}$ とする。このとき、バイナリマスク $\mathbf{B}\in\{0,1\}^{3\times4}$ を考える。このバイナリマスクにより以下を計算する。
>
> $$\mathbf{Z}=\mathbf{B}\odot\mathbf{X}+(\mathbf{1}-\mathbf{B})\odot\mathbf{Y} \tag{26}$$

この方式だと、出力の $\mathbf{Z}$ もある c チャンネル目についての値になるので注意してください。ここでは元の特徴マップを $\mathbf{X}\in\mathbb{R}^{2\times3\times4}$ として、c チャンネル目を $\mathbf{X}^c\in\mathbb{R}^{3\times4}$ とか $\mathbf{X}_c\in\mathbb{R}^{3\times4}$ としてもよいです。ですがそうすると式が複雑になります。複雑にしてでも正確に伝えたいか、あるいはそこは抽象化して本来強調したいところを強調すべきか、ケースバイケースですので、よく考えて決めるとよいと思います。

あるいは、マスクを複製する方式もあります。

> ある CNN の出力である 2 チャンネルの特徴マップを $\mathbf{X}\in\mathbb{R}^{2\times3\times4}$ とし、同様に別の CNN の特徴マップを $\mathbf{Y}\in\mathbb{R}^{2\times3\times4}$ とする。このとき、バイナリマスク $\mathbf{B}_0\in\{0,1\}^{3\times4}$ を考え、それをチャンネル方向に複製して結合したものを $\mathbf{B}\in\{0,1\}^{2\times3\times4}$ とする。このとき、次を考える。
>
> $$\mathbf{Z}=\mathbf{B}\odot\mathbf{X}+(\mathbf{1}-\mathbf{B})\odot\mathbf{Y} \tag{27}$$

この方式だと、$\mathbf{B}_0$ という別の変数を使ってしまいましたが、当初の表現にだいぶ近いです。

ポイントは、**10 年後に読まれても理解できる記述を心掛ける点**だと思います。「どうせみんな numpy を知っているからわかってもらえるだろう」とか、「流行っているフレームワークの記法なのでＯＫ」といった考えはやめましょう。研究とは知識を積み重ねることなので、いつ読まれても理解できるような客観的に正しい数式で書くことが重要です。numpy や pytorch が 10 年後にみんな読めるかはわかりません。

8　量化子は使わない

量化子（$\forall$ および $\exists$）を使いこなすことは難しいです。もし曖昧な気持ちで使っているのであれば、書かないことをオススメします。

9　どうしてもうまい書き方がわからないときは

もし説明したい内容を数式で記述することがどうしても難しければ、文章と図で説明してください。**中途半端に間違っている数式を書くことは逆効果**です。

10　その他、注意すべきこと

その他、注意すべき点を述べます。

- 変数や定数のために新しくアルファベットを導入したときは、必ず説明するようにしましょう。例えば $y=ax+b$ と書いたときは、y も a も x も b も説明しましょう。必要ないだろうと著者が変数の説明を省略することは、読者を混乱させる入口だと思います。
- 新しい概念を導入したり、一般的でない表記をする場合は、必ず初出のときに定義すると良いです。例えばコンピュータビジョン分野では、ノルム（$\|\mathbf{x}\|_2$）を導入した時でも、「これはユークリッドノルムです」と言っておくと親切でしょう。
- 慣例で決まっている記号は、それに従いましょう。例えば単位行列を $\mathbf{A}$ と定義することは不自然です。
- セクションをまたいだとしても、同じ記号を別の用途に使わないようにしましょう。例えばあるセクションで画像を $\mathbf{I}$ と定義した場合は、別のセクションで単位行列を $\mathbf{I}$ として使わないようにしましょう。
- tips ですが、arXiv の論文は tex のコードも公開されているため、それを読むことでお気に入りの論文の表記を真似することも出来ます。

11　チートシート

本資料で説明したポイントを表 2 にチートシートとしてまとめました。論文を書いたらこれを参考にチェックしてみるとよいです。

12　番外編：なぜロスは mathcal で引数無視なのか

近年の深層学習の論文では、ロス関数はなぜか mathcal で表記する上に引数を省略する傾向にあります。例えば

$$\mathcal{L}_{all}=\lambda_{adv}\mathcal{L}_{adv}+\lambda_{style}\mathcal{L}_{style} \tag{28}$$

のような式です。この表記がいつ始まりどのように広まったのか謎なのですが、これが慣習になってしまっています。これは、実際にコーディングする際は上記の抽象化で十分であるため、詳細は汲み取ってくれ、という表記になっています。慣習である以上従わざるをえないという面はあるので、実際に論文を書くときは近い関連研究を参考にしてください。

上の式をより正確に書くならば、例えば次のようにするべきなのかもしれません：訓練するパラメータを並べて $\boldsymbol{\theta}$ とする。それ以

推奨しない、または間違った記述	OK な記述	説明
ベクトルを x と書く	ベクトルを $\mathbf{x}$ あるいは $\boldsymbol{x}$ と書く	ベクトルは太字
$\mathbf{x} = [1, 2, 3]$	$\mathbf{x} = [1, 2, 3]^\top$	ベクトルは縦ベクトルと心に決める
D 次元のベクトル $\mathbf{x}$ を考える	D 次元のベクトル $\mathbf{x} \in \mathbb{R}^D$ を考える	変数はドメインを示す
上付き、下付きがたくさん：$x_{i,j}^k$	上付き、下付きを可能な限り減らす	上付き、下付きはわかりにくい
下付きの下付き：x_{a_i}	下付きの下付きはやめる	下付きの下付きは曖昧でわかりにくい
$\mathbf{x_i}$	$\mathbf{x}_i$	インデクスは太字にしない
$\mathbf{x}_{in}$	$\mathbf{x}_{\mathrm{in}}$	下付き上付きで二文字以上の単語を使うときはローマン
$score(x) + 10$	$s(x) + 10$	関数や変数に英単語を使わない
$a \in \mathbb{R}^3, b \in \mathbb{R}$ のときに $a + b$	$\mathbf{a} + b\mathbf{1}$	ブロードキャストをしない
要素積の意味で $\mathbf{AB}$	$\odot$ を定義して $\mathbf{A} \odot \mathbf{B}$	要素積はアダマール積
曖昧な気持ちで $\forall$ や $\exists$ を使っている	使うことをやめる	自信がなければ量化子は使わない

表 2　数式のチートシート

外のパラメータは定数だと考える。ここで、ロス関数は

$$L_{\mathrm{all}}(\boldsymbol{\theta}) = \lambda_{\mathrm{adv}} L_{\mathrm{adv}}(\boldsymbol{\theta}) + \lambda_{\mathrm{style}} L_{\mathrm{style}}(\boldsymbol{\theta}) \tag{29}$$

と定義される。