

Cookpad Summer Internship 2018

5 DAY R&D

機械学習理論と画像分析の講義資料

菊田 遥平 *

2018年8月26日

概要

この資料は Cookpad Summer Internship 2018 5DAY R&D^{*1} の講義資料のうち、機械学習理論と画像分析に関するものです。講義は他にも自然言語処理と ML Ops に関するものがありますが、それらは公開されたら別途まとめて共有します。

Internship の内容は講義だけでなく実データを使った分析なども含まれていますが、本資料では講義部分のみを切り出して公開しています。

この資料は是非色々な人に共有してもらいたいですが、後日修正する可能性もあるので、二次配布をせずに下記資料 URL を共有するようにしてください。

- Repository URL: <https://github.com/yoheikikuta/2018-cookpad-intern-yohei-lecture>

* Twitter: [@yohei_kikuta](#), Resume: [resume](#)
*1 <https://internship.cookpad.com/2018/summer/>

目次

1	機械学習理論：計算グラフと自動微分	3
1.1	計算グラフ	3
1.2	微分を主たる対象とする数値計算の手法	4
1.3	自動微分の基礎と具体的な計算方法	8
1.4	自動微分の機械学習への応用	10
1.5	演習	10
2	機械学習理論：deep learning の汎化性能	11
2.1	機械学習の汎化性能	11
2.2	VC 次元と Rademacher 複雑度	12
2.3	パラメタ数がデータ数よりも多い場合の汎化性能	14
2.4	学習率とミニバッチサイズで見る deep learning の汎化性能	16
2.5	演習	19
3	画像分析：クックパッドにおける活用事例と画像分析基礎	21
3.1	クックパッドにおける活用事例	21
3.2	画像分析の基礎	24
3.3	convolution の理解	28
3.4	演習	32
4	画像分析：軽量モデルの理解と実データを用いた演習	33
4.1	軽量なネットワークアーキテクチャの理解	33
4.2	実際の画像データを使った分析	37
付録 A	Appendix	38
A.1	テンソル	38
A.2	dual number による forward mode の定式化	39
A.3	パラメタ数がデータ数よりも大きな線形システム	39
A.4	物理学における Langevin 方程式	40

1 機械学習理論：計算グラフと自動微分

この章では計算グラフと自動微分について議論します。これらの要素は最近の deep learning ライブラリに欠かせない構成要素ですが、なかなか勉強する機会がない（と個人的に思っている）ので、このインターンで取り上げようと思いました。

この章は参考文献が比較的少ないため、最初にこの講義資料を作成する上で参考にした文献を挙げておきます。これらを十分に理解していればこの講義を聞く必要はありません。自分は講義を聞く必要性がないという人は、講義中は他の人に迷惑を掛けない範囲で何か好きなことを書いてください。

- DEEP LEARNING [1]

Goodfellow, Bengio, Courville による deep learning を包括的に取り扱っている書籍です。簡潔にではありますが 6.5.1 に計算グラフに関する記述があります。

- <http://www.cs.cornell.edu/courses/cs5740/2017sp/lectures/04-nn-compgraph.pdf>

Cornel 大の講義資料です。自然言語処理向けの講義ですが、基礎的なことが簡潔に書いてあるので最初にさっと目を通すには良い資料です。

- <https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf>

Stanford 大の講義資料です。TensorFlow [2] に特化した資料ではありますが、その分 TensorFlow の snippet と共に読み進めることができます。

- Automatic differentiation in machine learning: a survey [3]

自動微分を機械学習的な観点からサーベイした論文です。自動微分をある程度しっかり勉強したい人はこれを読み込むところから始めるのがよいと思います。今回の講義内容の多くもこの論文に基づいて構築されています。

1.1 計算グラフ

計算グラフとは、要求された計算に対する関数的な記述を表現することができる Directed Acyclic Graph (DAG^{*2}) です。計算をグラフによって表現する方法には色々なパターンが考えられますが、本資料では次の二つの要素から構成される DAG として定義します。

- ノード

テンソルの値を表現するもので、さらにテンソルに対する演算（関数の適用や四則演算など）もノードで表現します。

- エッジ

ノードに対するポインタの役目をするものです。テンソルの値の流れを表現するものだと考えてください。

本資料ではあまり数学的な取り扱いはせず、具体的な例などを基に理解する方向で議論を進めます。ただし、テンソルに関しては Appendix A.1 で簡単に説明しています。TensorFlow の登場により、数学的な対象であるテンソルが何者であるかを分かってないままテンソルという言葉を使うので混乱が生じる可能性があるためです。

ここからは、具体例として $p_\theta(x) = \text{sigmoid}(\tanh(Wx + b) + x)$ という予測確率に対するクロスエントロピー $f_\theta(x, y) = -y \log p_\theta(x) - (1 - y) \log(1 - p_\theta(x))$ を考え、計算グラフの議論をしていきます（図 1）。ここで、 x は入力データで y は教師ラベル、 W と b は学習対象となるパラメタです。まず、以下の記法を導入し

*2 本資料では DAG に関する詳しい議論は割愛します。Directed であるので向きがあり、Acyclic であるので環がないようなグラフのことです。DAG の特徴は色々ありますが、例えばトポロジカルソートができるというもので、依存関係のあるタスクを適切に表現することができます。deep learning で実行する演算を考えれば、DAG で表現できるということは同意できると思います。

ます。

- ノードには v という記号を使います。
- 入力ノードに関しては v_i の添字が非正整数 ($i \in \mathbb{N}, i \leq 0$) となるように設定し、これは入力データなどの決まった値 (TensorFlow でいうところの Placeholder) を与える役割もしくは重みやバイアスなどの学習パラメタ (TensorFlow でいうところの Variable) を与える役割、を担います。
- 中間ノードや出力ノードに関しては v_i の添字が正整数 ($i \in \mathbb{N}, i > 0$) となるように設定し、これは関数の適用や四則演算などの演算をする役割を担います。

ノードを全て書き下すと以下のようになります。

$$v_{-3} = y, \quad (1)$$

$$v_{-2} = x, \quad (2)$$

$$v_{-1} = W, \quad (3)$$

$$v_0 = b, \quad (4)$$

$$v_1 = v_{-2} \cdot v_{-1}, \quad (5)$$

$$v_2 = v_1 + v_0, \quad (6)$$

$$v_3 = \tanh(v_2), \quad (7)$$

$$v_4 = v_3 + v_{-2}, \quad (8)$$

$$v_5 = \text{sigmoid}(v_4), \quad (9)$$

$$v_6 = -v_{-3} \log v_5 - (1 - v_{-3}) \log(1 - v_5). \quad (10)$$

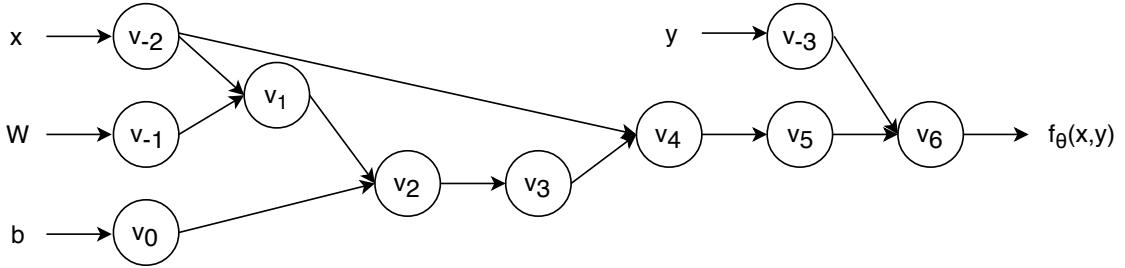


図 1 計算グラフの例。ここでは $f_\theta(x, y) = -y \log p_\theta(x) - (1 - y) \log(1 - p_\theta(x))$ where $p_\theta(x) = \text{sigmoid}(\tanh(Wx + b) + x)$ を考えています。

計算グラフを書き下せば、実際にどのようなステップで計算すればよいかは明瞭だと思います。DAG の流れに沿って、入力ノードに値を入れて各ノードで演算を適用していくことになります（後の節で具体的な数値計算をしてみます）。

機械学習で使う計算グラフを最低限理解することを目的とするならば、内容としてはこの程度で事足ります。ただし、重要な点はこの計算グラフを用いていかにして効率的に微分の値を計算するかです。それは現状の機械学習の学習アルゴリズムが、微分可能な目的関数を設定して目的関数の最適化をする、という枠組みを主流としているためです。それゆえに、以降では、計算グラフを使ってどのように関数の微分値を計算していくのか、に主眼を置いて議論を展開していきます。

1.2 微分を主たる対象とする数値計算の手法

上述のように、統計的機械学習における主たる対象の一つは目的関数の微分^{*3}です。この節では関数の微分値を数値的にどのように計算するかを列挙してその特徴を議論します。まず、微分を数値的に計算するには以下の4つの手法があります。

^{*3} 本資料では微分可能な関数のみを扱います。Rectified Liner Unit (ReLU) は微分不可能な点はあるのですが、話の都合上これも微分可能な関数であるとして同じ枠組みで扱います。もうちょっと真面目に言えば劣微分可能な関数のみを考える、ということですが本資料においてはそれほど本質的ではないです。

- 手動微分

人間が頑張って手で微分をするものです。やはり人手で微分をしたものは微係数の輝きが違う、とか何とか言いたいようなタイミングで使うことになるかもしれません。数値計算という観点では、手動微分によって解析的な微分の式が得られていればそこに数値を代入することで答えを得ることができます。当然のことながら、手動で対応するのは大変です。対象とする関数に対して事前に全て計算しておかねばならないですし、複雑な関数になれば手計算で対応するのは難しいです。

- シンボリック微分

式をコードに落としてから、その微分をプログラムで自動で実行するものです。数値的な計算という意味では手動微分と同じで計算した式に数値を代入するもので、手動微分の手動の部分をプログラムにやってもらうというイメージです。有名なソフトウェアとしては Mathematica [4] や Maple [5] や Maxima [6] があります。想像に難くないですが、人間がやるよりもうまくできる場合が多いので、例えば理論物理の人々は解析的な計算でこのようなソフトウェアを活用しています。自動で解析的に計算できればそれで問題ないわけですが、これも複雑な関数になると時間が掛かり過ぎるなどの問題があります。これは後に出てくる自動微分との区別において重要なポイントなので、後でもう少し詳しく議論します。

- 数値微分

数値微分は解析的な表現を介さずに数値的に処理をする手法です。関数 $f(\mathbf{x})$ の数値微分を実行するには naive には以下のようにすればよいです。

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \simeq \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad (11)$$

$$= \frac{\partial f(\mathbf{x})}{\partial x_i} + \mathcal{O}(h). \quad (12)$$

ここで \mathbf{e}_i は x_i 方向の単位ベクトルです。これは微分後の解析的な表現 ($f'(\mathbf{x})$ の表式) を必要としないのでその部分の計算のコストではなく、単に h の値を十分に小さくすることで微分値を取得することができます。ただし数学的には $h \rightarrow 0$ の極限を取ることで等式となるので、右辺の形で表現することで本質的に誤差を含んでいます（打切り誤差）。この誤差を小さくする簡単な方法として以下のように計算する方法があります。

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \simeq \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}, \quad (13)$$

$$= \frac{\partial f(\mathbf{x})}{\partial x_i} + \mathcal{O}(h^2). \quad (14)$$

これは Taylor 展開をすれば h の一次の項がキャンセルすることは明らかです。数値微分にはもう一つ誤差が入り込みます。差分を取ることによって桁落ちをして数値精度が悪くなるという誤差です（丸め誤差）。数値微分における打切り誤差と丸め誤差の関係はトレードオフになっています。図 2 がそれを示しているもので、 h の値を小さくすると分子の差をとったときに桁落ちが発生して誤差が大きくなり、逆に h を大きくすると近似式の精度が悪くなることにより打ち切り誤差が大きくなり精度が悪くなります。

- 自動微分

この自動微分が本節の主役となるものです。対象とする関数が \exp などの初等関数と四則演算の組み合わせから成ることを想定し、逐次的にそれらの初等関数や演算を適用することで対象とする関数を計算することを考えます。各ステップにおいて微分の表式を求めておいて^{*4}（これは微分の連鎖律の途中に出てくるものに対応します）、微分の値を計算する段階で実際に値を代入して計算する手法です。理解のポイントとなるのは、各ステップにおける微分の表式は closed form になっていないという点です

^{*4} この解析的な計算は事前に初等関数などの微分を計算したものを作成しておいて、それを組み合わせることで実装します。例えば TensorFlow では次のファイルに記述されています。
https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/math_grad.py

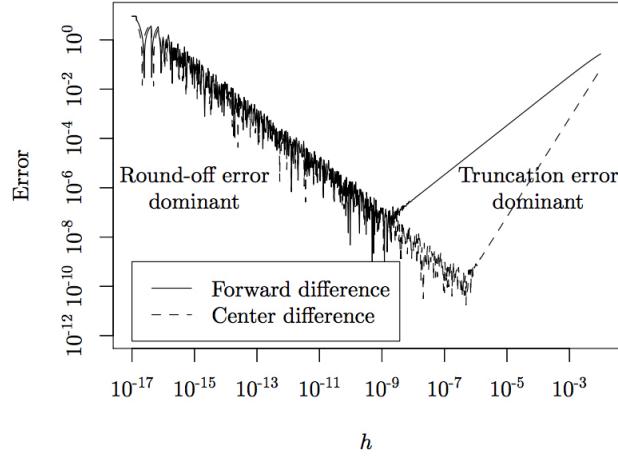


Figure 3: Error in the forward (Eq. 1) and center difference (Eq. 2) approximations as a function of step size h , for the derivative of the truncated logistic map $f(x) = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$. Plotted errors are computed using $E_{\text{forward}}(h, x_0) = \left| \frac{f(x_0+h)-f(x_0)}{h} - \frac{d}{dx}f(x)|_{x_0} \right|$ and $E_{\text{center}}(h, x_0) = \left| \frac{f(x_0+h)-f(x_0-h)}{2h} - \frac{d}{dx}f(x)|_{x_0} \right|$ at $x_0 = 0.2$.

図 2 数値微分における誤差. 図は [3] より引用.

(それゆえに多項式の整理などの複雑な計算は要求されません). このイメージを掴むために簡単な例として図 3 を見てみましょう. これは左から右に演算をしていくものになっていて、特に微分が連鎖律に

$$\begin{array}{ccc}
 \cancel{x^2} & e^{\cancel{x^2} + x^2} & (e^{\cancel{x^2} + x^2})^2 \\
 f(x) = x^2 & g(f(x)) = e^f + f & h(g(f(x))) = (g)^2 \\
 \frac{\partial f(x)}{\partial x} & \frac{\partial g(f)}{\partial x} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x} & \frac{\partial h}{\partial x} = \frac{\partial h}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{\partial x} \\
 2x & (e^f + 1) \cdot 2x & 2g \cdot (e^f + 1) \cdot 2x
 \end{array}$$

図 3 自動微分のイメージを掴むための例. 左から右への計算で、特に微分が連鎖律に従い前のステップの表式に依存した形で表現されています.

従い前のステップの表式で表現されています. 微分の最終的な形が x だけの言葉で書かれていないので closed form ではないですが、数値的に計算するならば各ステップでの計算結果を受け渡していくことで望む計算結果を得ることができます. 一方で各ステップでの微分の表式は（前のステップに依存はしていますが）解析的な式になっているので、数値計算をする上では打切り誤差を発生させずに微分値を算出することが可能となっています. このように、数値的な計算をする上では closed form であることは必要ないことと微分の連鎖律を用いて、うまくステップ毎に計算をする手法になっています. また、図 3 から計算グラフと相性が良さそうであることも推測できると思います. 自動微分は解析的な計算と数値的な計算のいいとこどりのような手法になっていますが、名前から実態が想像しにくいことに加えて、シンボリック微分との区別が混乱しがちなので、後に出てくる例などと合わせて違いをしっかりと理解してください.

言葉だけ眺めていても違いが分かりづらいかもしれませんので、これら 4 つの手法の違いを具体的な例を

ベースに理解してみましょう（図 4）。漸化式で計算できるような関数 $f(x)$ を考えます。これは例えば deep learningにおいては各層毎に演算をしていくことに対応しています。

- 手動微分

最終的な表式を解析的に計算してその結果をコーディングして、数値を代入することで数値微分を実現します。

- シンボリック微分

所与の式をコーディングし、そこからはプログラムにより解析的に微分演算を実行し、その後に数値を代入することで数値微分を実現します。

- 数値微分

$f(x)$ の closed form の式をコーディングした後、前述の数値微分をします。

- 自動微分

漸化式の各ステップで微分も計算しておいて、前後のステップに依存するような closed form でない式に基づいて、数値を代入して各ステップで値を受け渡していくことで数値微分を実現します。

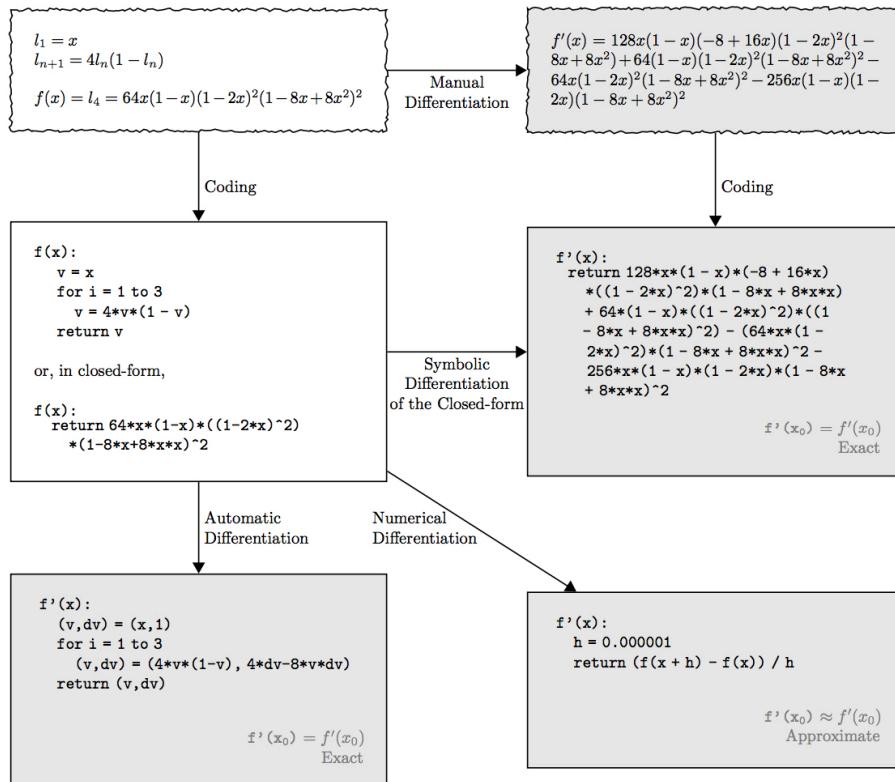


Figure 2: The range of approaches for differentiating mathematical expressions and computer code, looking at the example of a truncated logistic map (upper left). Symbolic differentiation (center right) gives exact results but requires closed-form input and suffers from expression swell; numerical differentiation (lower right) has problems of accuracy due to round-off and truncation errors; automatic differentiation (lower left) is as accurate as symbolic differentiation with only a constant factor of overhead and support for control flow.

図 4 各微分手法の違い。図は [3] より引用。

さらに手法の違いが実際の計算においてどう重要になるのかを考えてみます。手動微分は分かりやすいと思いますが、例えば deep learning では最終的な目的関数の表式からパラメタに関する微分を解析的に計算するのは、よほど強い制約を入れてモデルを構築しない限りは不可能です。当然ながら数値計算で使うことはほぼありません。数値微分に関しては前述の誤差の問題に加えて、入力次元と出力次元毎に関数を評価しなければ

ならないという問題も発生します。ですので、パラメタの次元が数百万や数千万にもなるような場合ではかなり大変となり、数値微分も主たる方法ではありません。しかし全く使われていないということではなく、解析的な計算が正しく機能しているかのチェックツール（gradient checking）として、解析的な計算と数値微分が一定の誤差の範囲でマッチしているかを確かめる用途で用いられたりしています^{*5}。

重要なのはシンボリック微分と自動微分の違いがどのような影響を及ぼすのかです。端的に言うと、式が複雑になるにつれてシンボリック微分の計算が大変になっていきます。もう少し具体的に言えば、例えば多項式の微分を考えるとき、closed form な表式を得ようとすると、多項式の次数に対してシンボリック計算は指数的に時間が掛かってしまいます（exponential swell）。これは微分の基本がライプニッツ則、 $\partial_x(f(x)g(x)) = \partial_x f(x)g(x) + f(x)\partial_x g(x)$ 、であることを考えれば、難しい理屈ではありません。closed form で解析的な表式が得られれば強力ですが、大量の演算によって一つのモデルが構築される deep learning などにおいては、これはあまり良い方向性とは言えません。

これに対して、自動微分は closed form な結果を得るような計算をするわけではないので exponential swell の問題は回避することができます。一方で各ステップでは解析的に微分を計算しているので、数値計算としては打ち切り誤差のない結果を得ることができます。繰り返しになりますが、実行したいのは数値計算なので closed form な表式を求めることはせず、微分の連鎖律をうまく使って各ステップでは前後のステップに依存する解析的な表式を求めて、その結果を使って数値計算をすることで打ち切り誤差を抑える、という賢い手法であることが分かります。ちなみに自動微分それ自体は古くからある技術で [7, 8]、大規模非線形システムの数値解析など様々な科学数値計算において使われている手法です。

ここまで自動微分のいくぶん抽象的な説明をしてきましたが、自動微分を使ってどうやって計算するかの具体的なイメージは持ててないと思いますので、以降では自動微分の具体的な計算と利点について議論していきたいと思います。

1.3 自動微分の基礎と具体的な計算方法

この節では自動微分の簡単な定式化と、重要な forward mode と reverse mode による微分値に関する具体的な例を用いて実際に計算をしてみます。forward mode と reverse mode とは自動微分を用いた計算方法の種類で、特に機械学習のように入力の次元が出力の次元よりも大きい場合に reverse mode が効率的な計算になっていることを確認します。

まずは forward mode から考えていきます。中間ノード $v_i (i \in \mathbb{N}, i > 0)$ に対する v_0 による微分を次のように書くことにします。

$$\dot{v}_i = \frac{\partial v_i}{\partial v_0}. \quad (15)$$

ここで、 v_0 は微分値を求めるべきパラメタに対応する変数です。最終的な目的関数の微分を求めるためには、微分の連鎖律を考えれば、 $\dot{v}_{i+1} = \frac{\partial v_{i+1}}{\partial v_i} \frac{\partial v_i}{\partial v_0}$ のように前のステップの計算結果を使って新しいステップの計算を繰り返していくことが分かります。いま、関数 $l: \mathbb{R}^n \rightarrow \mathbb{R}^m$ のヤコビアンを計算することを考えます。これは（一般に複数の）目的関数を入力変数で微分するという意味で、まさに我々が計算したい対象です。

$$\mathbf{J}_l = \left[\begin{array}{ccc} \frac{\partial f_1}{\partial v_{-(n-1)}} & \cdots & \frac{\partial f_1}{\partial v_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial v_{-(n-1)}} & \cdots & \frac{\partial f_m}{\partial v_0} \end{array} \right] \Big|_{v_- = \mathbf{a}}. \quad (16)$$

ここで、 \mathbf{v}_- は $v_k (k = -(n-1), \dots, 0)$ を各要素とするベクトルで、 \mathbf{a} は微分を評価する点です。式 (16) の 1 列を計算することを考えます。これは入力変数 \dot{v}_{-k} を一つ定めて $f_j (j = 1, \dots, m)$ の微分を計算することに

^{*5} TensorFlow であれば https://github.com/tensorflow/tensorflow/blob/r1.9/tensorflow/python/ops/gradient_checker.pyなどを参照してください。

なります。

$$\dot{f}_j = \left. \frac{\partial f_j}{\partial v_{-k}} \right|_{v_- = a} . \quad (17)$$

計算グラフの観点からは, v_{-k} のうち一つだけ 1 におき他のものは 0 として, 各ステップの計算を実施すればよいことになります。重要な点として, 入力変数を固定して一回計算グラフの前ステップを計算していくべき, 全ての f_j に対する結果を得ることができます（これは計算グラフの性質から明らかです）。一回の計算で一列列計算できるので, あとは n 列全部に関して繰り返してやればよいことになります。以上により, この場合はヤコビアンの計算に n 回の iteration が必要となることが分かります。機械学習でよく使われるのは f がスカラーの場合 ($m = 1$) で, この場合はヤコビアンが $1 \times n$ のベクトルだと考えることができますが, 結局 n 回 iteration しなければならないということは変わりません。逆に言えば, $n \gg m$ の場合, この計算は効率的なものとなりますが, 機械学習ではあまり遭遇することがないケースです。

具体的に手計算をすることで感覚を掴むことにしましょう。対象とするのは図 1 の計算グラフです。入力変数の値を適当に定めて, 適当な有効数字で各ステップの計算を実際に実施してみましょう。forward modeにおいては微分値の計算も並行して進めていくことができる, パラメタ W を対象として, 図 1 と比べながら微分値の計算も並行して実施してみましょう。

Forward Primal	Forward Derivative
$v_3 = x = 1$ $v_2 = x = 0.5$ $v_1 = W = 3$ $v_0 = b = -1$	$\dot{v}_3 = \dot{x} = 0$ $\dot{v}_2 = \dot{x} = 0$ $\dot{v}_1 = \dot{W} = 1$ $\dot{v}_0 = \dot{b} = 0$
$v_1 = v_2 \cdot v_1 = 0.5 \cdot 3 \quad 1.50$	$\dot{v}_1 = \dot{v}_2 v_1 + v_2 \dot{v}_1 = 0.5 \cdot 1 \quad 0.500$
$v_2 = v_1 + v_0 = 1.5 - 1 \quad 0.500$	$\dot{v}_2 = \dot{v}_1 + \dot{v}_0 = 0.5 \quad 0.500$
$v_3 = \tanh(v_2) = \tanh(0.5) \quad 0.462$	$\dot{v}_3 = \operatorname{sech}^2(v_2) \cdot \dot{v}_2 = \operatorname{sech}^2(0.5) \cdot 0.5 \quad 0.393$
$v_4 = v_3 + v_2 = 0.462 + 0.5 \quad 0.962$	$\dot{v}_4 = \dot{v}_3 + \dot{v}_2 = 0.393 \quad 0.393$
$v_5 = \text{sigmoid}(v_4) = \text{sigmoid}(0.962) \quad 0.724$	$\dot{v}_5 = \frac{e^{-v_4}}{(1+e^{-v_4})^2} \cdot \dot{v}_4 = \frac{e^{-0.962}}{(1+e^{-0.962})^2} \cdot 0.393 \quad 0.0786$
$v_b = -v_3 \ln v_5 - (1-v_3) \ln(1-v_5)$ $= -\ln 0.724 \quad 0.323$	$\dot{v}_b = -v_3 \ln v_5 - v_3 \cdot \frac{\dot{v}_5}{v_5} + v_3 \ln(1-v_5) - (1-v_3) \cdot \frac{\dot{v}_5}{1-v_5} = -\frac{0.0786}{0.724} - 0.109$
$f_0(x, y) = 0.323$	$\frac{\partial f_0(x, y)}{\partial W} = \dot{v}_b = -0.109$

図 5 forward mode での具体的な計算例。これらの計算は各ステップで並行して進めていくことができます。

ここで計算した結果が妥当であるかどうかはどう判断すればいいでしょうか。定性的には, パラメタの更新式が $W \rightarrow W - \eta \frac{\partial f}{\partial W} (\eta > 0)$ なので, この結果は W を大きくする方向に働き, これは v_5 を 1 に近づけるように寄与するので正しそうである, と理解できます。もう一つのパラメタ b に関しても同様に計算してみてください。

数学的には dual number を用いた forward mode の定式化をすることもできます。ここでは解説しませんが, Appendix A.2 で軽く説明していますので, 興味がある人は読んでみてください。

続いて機械学習で重要な reverse mode について説明します。名前から想像できるように逆から辿って

いくことになるので、出力ノードに対する中間ノードによる微分を考えます（adjoint な微分と呼ぶ）。

$$\bar{v}_i = \frac{\partial f_j}{\partial v_i}. \quad (18)$$

一つ前のステップの計算をしたければ、微分の連鎖律を用いて以下のように計算することができます。

$$\bar{v}_{i-1} = \frac{\partial f_j}{\partial v_{i-1}} = \frac{\partial f_j}{\partial v_i} \frac{\partial v_i}{\partial v_{i-1}} = \bar{v}_i \frac{\partial v_i}{\partial v_{i-1}}. \quad (19)$$

基本的には forward mode とは逆の順番で微分の連鎖律を辿っていけばよいですが、いくつか注意すべき点があります。まず、出力ノードから微分値を計算していくので、 $\mathbf{v}_- = \mathbf{a}$ に対する（出力ノードに近い）中間ノードの値が必要になります。それゆえに forward mode と違って並行に計算していくことはできず、まずは最初の手順として $\mathbf{v}_- = \mathbf{a}$ に対する全ノードの値を計算して保持しておく必要があります。次の手順で adjoint 微分を使って出力から入力へと辿っていくことになります。必要になる計算回数にも注意が必要です。出力から辿っていくので、 $\bar{f}_j = 1$ として他の f は全部 0 として計算していくので、これも forward mode と逆になっています。すなわち、一回の計算で全入力ノードへと伝播していくので、対象とする一つの f に対する求めたいパラメタの微分値は一回の計算で得ることができます。式 (16) の言葉で言えば、一回の計算で一行計算ができることがあります。それゆえに reverse mode では $n \ll m$ の場合に効率的な計算となり、機械学習のようにパラメタ数は多いが少數の目的関数から成るような場合で特に有用となります。

言葉の説明だけでは理解が難しいと思いますので、reverse mode でも同じ例で具体的に手計算してみましょう。図 1 の計算グラフとよく見比べながら微分の連鎖律の計算をしていきましょう。こちらはまず入力変数を固定して一度 forward 方向に全ステップを計算して中間ノードの値を保持しておく必要があります。次に微分に計算した中間ノードの値を用いて、reverse 方向への計算をしていきましょう。計算ができたら、計算結果が forward mode と合致しているか確認してみましょう。

1.4 自動微分の機械学習への応用

deep learning を取り扱うライブラリにおいては自動微分は必要不可欠な構成要素となっています。しかしながら、どのタイミングで計算グラフを構築し、またそれを評価するか、というのはライブラリによって異なります。define-and-run というのが前もって計算グラフを構築し、その後評価を実施するもので、TensorFlow や Caffe [9] などがこれにあたります。define-by-run というのが計算グラフの構築と評価を同時に実施するもので、PyTorch [10] や Chainer [11] などがこれにあたります。前者はコンパイルにより評価を高速化できるという利点や後者は必要に応じて柔軟にモデルを構築できるという利点があります。本資料では詳しくは議論しないので、興味がある人は調べてみてください。

1.5 演習

文中でも述べているように、図 5 と図 6 の例を自分自身の手で計算してください。計算結果と本資料を比較し、資料が間違っている場合は教えてください。

Forward Primal

$$\begin{aligned} v_3 &= z = 1 \\ v_2 &= x = 0.5 \\ v_1 &= w = 3 \\ v_0 &= b = -1 \end{aligned}$$

$$v_1 = v_2 \cdot v_1 = 0.5 \cdot 0.3 \quad 1.50$$

$$v_2 = v_1 + v_0 = 1.5 - 1 \quad 0.500$$

$$v_3 = \tanh(v_2) = \tanh(0.5) \quad 0.462$$

$$v_4 = v_3 + v_2 = 0.462 + 0.5 \quad 0.962$$

$$v_5 = \text{sigmoid}(v_4) = \text{sigmoid}(0.962) \quad 0.724$$

$$v_b = -v_3 \ln v_5 - (1-v_3) \ln(1-v_5) \\ = -\ln 0.724 \quad 0.323$$

$$f_0(x, y) = 0.323$$

Reverse Derivative

$$\begin{aligned} \bar{z} &= \overline{v_3} \\ \bar{x} &= \overline{v_2} \end{aligned} \quad \left. \begin{array}{l} \text{placeholders} \\ \vdots \end{array} \right.$$

$$\bar{w} = \overline{v_1} = -0.109$$

$$\bar{b} = \overline{v_0} = -0.217$$

$$\bar{v}_1 = \overline{v_1} \frac{\partial v_1}{\partial v_2} = \overline{v_1} \cdot v_2 = -0.109.$$

$$\bar{v}_2 = \overline{v_2} \frac{\partial v_2}{\partial v_0} = \overline{v_2} \cdot 1 = -0.217$$

$$\bar{v}_1 = \overline{v_1} \frac{\partial v_1}{\partial v_1} = \overline{v_1} \cdot 1 = -0.217$$

$$\bar{v}_3 = \overline{v_3} \frac{\partial v_3}{\partial v_2} = \overline{v_3} (1 - \tanh^2(v_2)) = -0.217$$

$$\bar{v}_3 = \overline{v_4} \frac{\partial v_4}{\partial v_3} = \overline{v_4} \cdot 1 = -0.217$$

$$\bar{v}_4 = \overline{v_5} \frac{\partial v_5}{\partial v_4} = \overline{v_5} \frac{e^{-v_4}}{(1 + e^{-v_4})^2} = -0.217$$

$$\bar{v}_5 = \overline{v_6} \frac{\partial v_6}{\partial v_5} = \overline{v_6} \left(\frac{v_3}{v_5} - \frac{1-v_3}{1-v_5} \right) = -1.38$$

$$\overline{v_6} = \overline{f_0(x, y)} = 1.$$

図 6 reverse mode での具体的な計算例. 計算は二つのステップから成り, 最初のステップでは入力値に対して各ノードの値を計算して保持し, 次のステップで出力ノードから微分の連鎖律を辿って入力ノードの微分値を計算します.

2 機械学習理論 : deep learning の汎化性能

この章では deep learning の有する汎化性能に関して議論します. これは広範に渡る話題なので, 本資料では講師の好きなごく一部のトピックのみを扱います. まず, 汎化性能とはそもそも何であるかという話と, 汎化性能の議論でよく使われる Vapnik Chervonenkis (VC) 次元 [12] と Rademacher 複雑度の話をします^{*6}. その後, deep learning に代表されるようにパラメタ数が非常に大きい場合に, そのような既存の枠組みでは汎化性能を議論しきれないという事実を議論します. 最後に deep learning の持つ解の性質と汎化性能の関係について最近の発展の一部を紹介します.

2.1 機械学習の汎化性能

問題設定として, 入力 x に対してラベル y を予測する教師有り学習を考えます. モデルとして $h \in \mathcal{H}$ を考えます. この \mathcal{H} はある条件を満足するような関数空間の元から成る集合で, このある条件というのは問題設定によって変えるものとなります. 学習データ数が N 個あり, 損失関数を $l(y, h(x))$ とすると, 訓練誤差最小化は形式的には次のように定義できます.

$$\min_{h \in \mathcal{H}} \hat{L}(h) = \min_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^N l(y_i, h(x_i)). \quad (20)$$

^{*6} 本来はこの辺りの議論は様々な数学的な背景知識が必要になりますが, 本資料ではあまり深くは立ち入らずに議論します.

典型的には、モデル h をかなり限定された関数空間の元として、その関数を特徴づけるパラメタ θ に対して損失関数が最小になるようにパラメタを調整する、ということをします。探索する関数空間を限定すればこの式(20)を最小化すること自体は難しいことではありません。実際、パラメタ θ に十分な自由度があれば任意に小さな誤差を出すようなモデルを作ることが可能です。

しかし、多くの場合で我々がやりたいことは、同時確率分布 $P(X, Y)$ からサンプルされるどんなデータに対しても高い性能を発揮することのできるモデルです。すなわち、以下の損失の期待値を最小化することです。

$$L(h) = \mathbb{E}_{P(X, Y)}[l(Y, h(X))] = \int dP(X, Y)l(Y, h(X)). \quad (21)$$

この値が小さいモデルであるほど汎化性能が高いモデルであると言います。これは先ほどとは異なり非常に難しい問題です。もしこの宇宙をすべて記述できる神ならばこの $P(X, Y)$ は知り得るものかもしれません、我々人間には一般に知り得ないものです。語り得ぬものには沈黙しなければならないと言いたくありますが、注意深く議論を進めることで、訓練誤差を最小化するモデルと汎化誤差を最小化するモデルの関係性を調べることができます。以降では少し記法が複雑になるので、最初に整理しておきます。

\tilde{h} : 式(21)を現実的に考慮できる関数^{*7}の中で最小化する元

h^* : 式(21)をモデル集合 \mathcal{H} の中で最小化する元

\hat{h} : 式(20)を解くことで得られる、訓練誤差を最小化する元

これらを使って、相対的汎化誤差は次のように書きます。

$$L(\hat{h}) - L(\tilde{h}) = (L(\hat{h}) - L(h^*)) + (L(h^*) - L(\tilde{h})). \quad (22)$$

ここで、 L は \hat{L} ではないのでどれも同時確率分布の下での損失期待値であることに注意してください。 $L(\hat{h}) - L(h^*)$ は訓練誤差を最小化するモデルが達成できる限界誤差で、 $L(h^*) - L(\tilde{h})$ が我々が考慮するモデル集合の元が達成できる限界誤差です。これは我々のモデル集合の選択に依存するので、モデルバイアスとも呼ばれます。第二項はモデル集合を十分に広く取ることでゼロにすることができます。しかし、第二項をゼロにするような大きなモデル集合を選ぶのが良い選択なのかを知るには、第一項に与える影響を調べる必要があります。モデル集合を大きく選ぶことで手元のデータセットに過学習した \hat{h} となってしまい、それゆえに直感的には $L(\hat{h})$ が大きくなることが分かります。より詳しく関係性を調べるために、第一項を次のように分解します。

$$L(\hat{h}) - L(h^*) = (L(\hat{h}) - \hat{L}(\hat{h})) + (\hat{L}(\hat{h}) - \hat{L}(h^*)) + (\hat{L}(h^*) - L(h^*)) \quad (23)$$

各項の意味を考えてみます。

第一項：訓練データで決まる \hat{h} が汎化誤差をどれくらい小さくできるかに依存

第二項： \hat{h} が訓練誤差を最小化するという定義なので非正

第三項：大数の法則などから確率オーダー $\leq O_p\left(\frac{1}{\sqrt{N}}\right)$ ^{*8}

第二項は非正で、十分なデータ数があれば第三項も小さい値とすることができます。問題となるのは第一項です。これは訓練データから構築される \hat{h} なので訓練データに依存していて、大数の法則などを用いた解析が適用できません。この部分の解析のために様々な手法が発展していますが、以降では特に有名な VC 次元と Rademacher 複雑度について見ていきます。

2.2 VC 次元と Rademacher 複雑度

簡単のため二値判別の場合を考えます： $h(x) \in \{\pm 1\}$ 。VC 次元も Rademacher 複雑度もある種のモデルの表現力や複雑さを測るような指標になっています。

*7 ここでは可測関数を指します。そもそも可測集合とは空集合の測度が 0 で完全加法性を持つ測度が定義される完全加法族です。長い話になるので、興味がある人は自分で調べてください。我々が通常取り扱う関数は可測関数であると思っておけばよいです。

*8 ここでは証明には立ち入りません。真面目に勉強したい方は [13]などを読んでください。

まずは VC 次元を定義します。モデル集合の元はデータ集合 \mathcal{X} の元を $\{\pm 1\}$ に写す写像ですが、データ集合を手元に得られた学習データ $X \subset \mathcal{X}$ に限定して、 \mathcal{H} を X に「制限」することを考えます。この「制限」とは、 X から $\{\pm 1\}$ への写像をモデルによって実施して、その写像により得られる元で集合を作ることでモデルを表現することを指します。もう少し具体的に言えば、 h を $x_i \in X$ に適用して得られる元を集めて集合を作ればよいことになります。

$$S_X(\mathcal{H}) = \{h(x_1), \dots, h(x_N) | h \in \mathcal{H}\}. \quad (24)$$

この集合は、各データ点を $\{\pm 1\}$ に写す可能な組み合わせをモデル $h \in \mathcal{H}$ がどれくらい尽くせるかというものになっています。この集合を用いてシャッターという概念を導入します。これは

$$|S_X(\mathcal{H})| = 2^{|X|}. \quad (25)$$

を満たす場合で、モデル集合 \mathcal{H} の X への制限が X から $\{\pm 1\}$ へ写す関数の全ての集合であることを意味しています。このシャッターを用いて VC 次元を定義します。 \mathcal{H} の VC 次元は $X \subset \mathcal{X}$ のうちシャッターできる最大の $|X|$ で定義されます。これは少し理解が難しいかもしれませんので例を考えてみましょう。 \mathcal{H} が線型モデルで \mathcal{X} が二次元平面の点全体、 X が二次元平面のいくつかの点であるという状況を考えます。与えられた X の点を $\{\pm 1\}$ に写すことを考えれば、線形モデルによってこれらの点の分離を考えることになります。

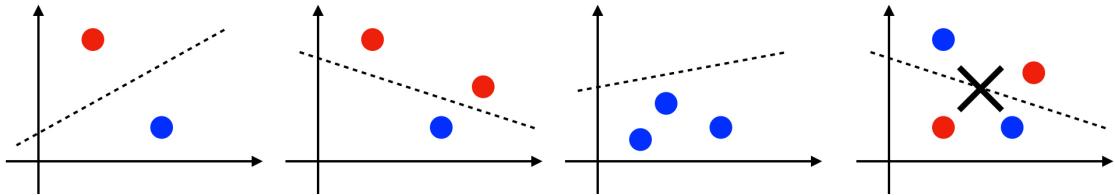


図 7 VC 次元のためのシャッターの例。赤丸と青丸がそれぞれ $+1$ と -1 のラベルであることを意味しています。点の数が 4 つ以上になると一般に線型分離が不可能になります。

図 7 を見れば、直感的にも理解しやすいと思います。データ点が 3 つ以下の場合は各点をそれぞれ $\{\pm 1\}$ に写す 2^3 個の場合が一つの線形モデル h で実現できます。一方でデータ点の数が 4 以上になると線型分離不可能であり、線形モデルにおいては 2^4 個の場合は表現できません。それゆえに、線形モデルの二次元平面の点における VC 次元は 3 となります。同様の問題設定で、モデルを円の内外で分離するようなものを考えた場合、VC 次元がいくつになるか計算してみてください。ここまで議論で、VC 次元がモデルの表現力を表していることが理解できると思います。また、モデルを複雑にすれば VC 次元を大きくすることができますが、これが複雑さとも関係があります。

次に Rademacher 複雑度を定義します。確率変数として $\sigma_i (i = 1, \dots, N), P(\sigma_i = \pm 1) = \frac{1}{2}$ を導入することで、Rademacher 複雑度を次のように定義できます。

$$R_N(\mathcal{H}) = \mathbb{E}_\sigma \left[\sup_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \sigma_i h(x_i) \right]. \quad (26)$$

これも定義だけでは理解するのが少し難しいので、例を考えてみます。 $N = 2$ の場合を考えます。モデルが返す値が $\{h(x_1) = 1, h(x_2) = 1\}$ のみである（考へているモデル集合のどのモデルを使ってもこれしか返せない）場合を考えると、

$$R_2(\mathcal{H}') = \frac{1}{2} \left((1+1) + (-1+1) + (1-1) + (-1-1) \right) = 0, \quad (27)$$

という結果が得られます（モデル集合にはプライムをつけて区別しています）。これはモデルの返す値は一意なので \sup はなくなり、 $\sigma_i (i = 1, 2)$ がそれぞれ $\frac{1}{2}$ の値を取る場合の期待値を計算していることに注意してください。この結果が意味することは、モデルは単一の結果しか返さないのでこのモデル空間は複雑さがないと

いうことです。もう少し複雑な場合として $\{(h(x_1) = 1, h(x_2) = 1), (h(x_1) = -1, h(x_2) = -1)\}$ である場合を考えてみましょう。今度は

$$\begin{aligned} R_2(\mathcal{H}'') &= \frac{1}{2} \frac{1}{4} (\max(1+1, -1-1) + \max(-1+1, 1-1) + \max(1-1, -1+1) + \max(-1-1, 1+1)), \\ &= \frac{1}{2}, \end{aligned} \quad (28)$$

という結果が得られます（モデル集合にはプライムを二つけて区別しています）。今度はモデル空間の複雑さが増したために値が大きくなっています。モデル空間を拡張して $N = 2$ の全ての場合を尽くすときに値が 1 になることを確認しましょう（この事実は後で使います）。Rademacher 複雑度の解釈として、ランダムラベルに当てはめることができる性能と考えることができます。モデル空間が可能なラベルの組み合わせ（それはランダムにラベルを付与することで実現できます）をどれくらい尽くせるかを測っているためです。直感的にはどんなランダムラベルにもフィットできるようなモデルは、過学習をしているものなので複雑過ぎるモデルであると考えることができます。

これらの道具を使うことで、式 (23) の右辺第一項に関する議論を進めることができます。ここからは代表的な結果のみを紹介し、詳細には立ち入らないことにします。詳しい議論を追いたい方は [13]などを読んで下さい。ある条件の下で、

$$L(\hat{h}) - \hat{L}(\hat{h}) \leq C \left(R_N(\mathcal{H}) + \frac{\log(1/\delta)}{N} \right), \quad (29)$$

が確率 $1 - \delta$ で成り立つことが知られています。また、 C はある定数です。これはモデルが複雑過ぎると汎化性能（の上限）が悪くなってしまうことを意味しています。この Rademacher 複雑度を測るために VC 次元を利用することができます。

$$R_N(\mathcal{H}) \leq C' \sqrt{\frac{V_{\mathcal{H}}}{N}}, \quad (30)$$

ここで、 C' はある定数です。これにより、

- モデル集合を拡張して訓練誤差を 0 にするような複雑なモデルを許容するようにモデル集合を広げる
- VC 次元が大きい表現力の高い複雑なモデルになる
- Rademacher 複雑度も大きくなる
- 結果として式 (23) の右辺第一項（の上限）も大きくなる

という帰結が得られます。定数依存性などもあるため、どれくらいモデル集合を広く取るのがちょうどよいかというのは問題設定（データやモデル）に依存しますが、傾向としては確かに単純に広いクラスのモデル集合を考えれば良いというわけではないことを理解することができます。

ここでは詳細な議論をスキップしているので気になる人は自分で調べてほしいですが、メッセージとしては「あまりに複雑過ぎるモデルは汎化性能を高めるのに良くない」というものです。これは長い間支持されてきた考えた方で、データの特徴を捉えつつモデルを簡潔にすることが汎化性能を高めるのに重要だと思われてきました。その他にも Akaike Information Criterion (AIC) [14, 15] なども参照してください。

2.3 パラメタ数がデータ数よりも多い場合の汎化性能

先の議論に基づけば、パラメタ数が多い場合はモデルが複雑になりがちで汎化性能の観点では好ましくないようと思われます。特に Neural Network は比較的パラメタ数が多くなりがちなので、学習の難しさも相まって、汎化性能の高いモデルを作るという観点では有力ではないものでした^{*9}。

deep learning の理論的な表現力に関して少し考えてみます。ご存知の通り、Neural Network には普遍性定理 [16] があります。これは中間層が一層の Neural Network が（十分な数のノードがあれば）任意の連続関数

^{*9} 例えば Neural Information Processing Systems (NIPS) 2015 で Tibshirani 氏が「昔は、Neural Network はパラメタ数がデータ数よりも多い状況でフィッティングしようとして馬鹿げてる、みたいに言われていた」という話をしていました。

を任意の精度で近似できるというものです。この発展形として、ReLU を使う場合 [17] や有限のデータ点の場合 [18] など様々ありますが、重要なのは deep learning は学習データに完全にフィットできるようなモデルであるということです。これは過学習と思われる汎化性能の観点では好ましくないように感じられます。

しかしながら、deep learning が ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 で衝撃的な結果 [19] を残し、その後も高い汎化性能を発揮するモデルが次々と提案されていることから、パラメタ数が多いが汎化性能も高いという特徴が興味ある研究対象となっていました。実際に様々な研究がなされていますが、比較的最近で興味深いものとして deep learning の理解と汎化性能に関して一石を投じた論文 [18] に注目してその内容を簡単に紹介したいと思います。

まず、deep learning を真のラベルから一定割合でランダムなラベルに置き換えた randomization test の結果（図 8）を見てみましょう。

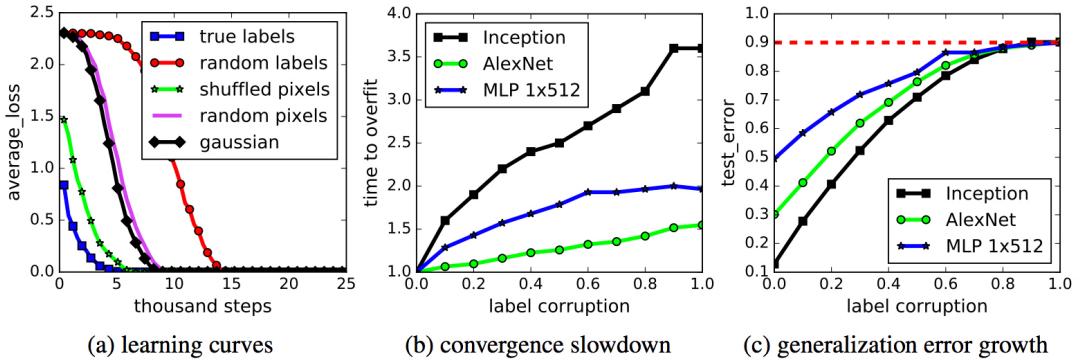


Figure 1: Fitting random labels and random pixels on CIFAR10. (a) shows the training loss of various experiment settings decaying with the training steps. (b) shows the relative convergence time with different label corruption ratio. (c) shows the test error (also the generalization error since training error is 0) under different label corruptions.

図 8 randomization test の結果。図は [18] より引用。

この結果から、deep learning はランダムなラベルであっても完全にフィットできる、しかし真のラベルを与えた方が収束が速い、真のラベルが多い方が汎化性能が高い、ということが見て取れます。すなわち、deep learning は確かに過学習し切るだけの表現力はあるが、データを単純に暗記しているのではなく、何か意味ある特徴を学んでいてそれが汎化性能にも役立つべきだと推測できます。この辺りの解析は deep learning の memorization [20] でもなされているので合わせて読んでみてください。

汎化性能には正則化も重要です。正則化には explicit なもの (weight decay など) と implicit なもの (batch normalization や augmentation など) があり、それらの効果を調べたのが図 9 になります。

この結果から、どちらのタイプの正則化も確かに効いているが、正則化なしでも高い汎化性能を誇っていることから、汎化性能は正則化だけでは説明できないことが分かります。

実はここで述べているようなパラメタ数がデータ数よりも多い場合というの deep learning に限らず線形システムにおいても非自明な結果を与えます。これは Appendix A.3 に書きましたので、興味がある人は読んでみてください。

汎化性能の指標の話に戻ってみましょう。これまでの観察から、deep learning に対しては前節で用いた VC 次元や Rademacher 複雑度が適用できないことが分かります。VC 次元は全データ点を分割することができる所以大きな値^{*10}となり、Rademacher 複雑度はランダムなラベルにフィットできるので 1 となります。それでも deep learning は高い汎化性能を示します。そこで、deep learning の汎化性能はどのような指標で測れるかということが疑問として湧きますが、これに関しては決定的なものはありません。例えばこの論文 [21] では、重みの path に沿った和や重みに摂動を加えた場合の robustness に基づく sharpness (とその拡張) な

^{*10} 任意に大きいサイズのシャッターが可能な場合は無限大の VC 次元と呼びます。

Table 1: The training and test accuracy (in percentage) of various models on the CIFAR10 dataset. Performance with and without data augmentation and weight decay are compared. The results of fitting random labels are also included.

model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75
		(fitting random labels)	no	100.0	9.78
Inception w/o BatchNorm	1,649,402	no	yes	100.0	83.00
(fitting random labels)	1,387,786	no	no	100.0	82.00
		no	no	100.0	10.12
		yes	yes	99.90	81.22
		yes	no	99.82	79.66
		no	yes	100.0	77.36
Alexnet	1,387,786	no	no	100.0	76.07
		no	no	99.82	9.86
MLP 3x512	1,735,178	no	yes	100.0	53.35
		no	no	100.0	52.39
		(fitting random labels)	no	100.0	10.48
MLP 1x512	1,209,866	no	yes	99.80	50.39
		no	no	100.0	50.51
		(fitting random labels)	no	99.34	10.61

図 9 正則化の有無による学習データとテストデータの正答率の結果. 図は [18] より引用.

ど、様々な指標を実験的に調べていますので興味がある人は読んでみてください. deep learning の汎化性能の理解は盛んに調べられている研究対象の一つです.

以上でパラメタ数がデータ数よりも多い場合の汎化性能の話をしましたが、実用上どのように deep learning を取り扱えばより高い汎化性能が得られるのかというのも重要な問題です。実務においては実績のあるモデルや様々な augmentation を駆使するのが良いかと思います^{*11} が、理論的にはなぜ勾配法のように局所解に陥りやすい学習アルゴリズムで良い汎化性能を有する解に辿り着けるのかという疑問が残ります。

例えば linear deep neural network では二乗誤差の場合は全ての局所解が大域解と等しくなることが示されています [22]. 非線形性を入れると大域解を解析的に求めるというのは難しくなりますが、性質として解の周りの曲率が小さい正の値となる flat な解の方が曲率の大きい sharp な解よりも良い汎化性能を示すことが様々な研究（古くは [23] など）で報告されています^{*12}. 実験的な観点からは flat な解を見つけるため特に learning rate や mini-batch サイズに注目して色々な研究がなされました（例えば [25, 26] など）. 次節では、learning rate と mini-batch サイズがどのように flat な解と関係づくのか、について解析した論文 [27] について詳しく解説します。

2.4 学習率とミニバッチサイズで見る deep learning の汎化性能

まず、簡単な場合として 1 次元パラメタ ω を考えます。

$$P(\omega|y, x; M) = \frac{P(y|\omega, x; M)P(\omega; M)}{P(y|x; M)}. \quad (31)$$

^{*11} 様々な分野の State Of The Art (SOTA) のモデルを常に把握しておくというのは大変ですが、例えば SOTA のモデルを集めた GitHub レポジトリ のようなものもあります。

^{*12} 実は sharp な解でも良い汎化性能を示すことが報告されています [24]. 一般的な性質を議論することが難しいことを思い知らされますが、ここではこの話には立ち入らないことにします。

ここで, x は入力, y はラベル, M はモデルを表します. この式の尤度を exponentiate して以下のように書き直します.

$$\begin{aligned} P(y|\omega, x; M) &= \prod_i P(y_i|\omega, x_i; M), \\ &= e^{-H(\omega; M)}. \end{aligned} \quad (32)$$

ここで, $H(\omega; M) = -\sum_i \ln P(y_i|\omega, x_i; M)$ としました. gaussian prior $P(\omega; M) = \text{sqrt}\frac{\lambda}{2\pi}e^{-\frac{\lambda\omega^2}{2}}$ を考えれば, 未知ラベル y_t の予測は式 (31) を用いて次式で書けます.

$$P(y_t|x_t, x, y; M) = P(y_t|\omega, x_t; M) P(\omega|y, x; M), \quad (33)$$

$$= \frac{\int d\omega P(y_t|\omega, x_t; M) e^{-C(\omega; M)}}{\int d\omega e^{-C(\omega; M)}}, \quad (34)$$

ここで, $C(\omega; M) = H(\omega; M) + (\lambda\omega^2)/2$ としました. 分母は確立の規格化を意識して式 (31) からさらに具体化して書いています. モデル比較はデータを所与とする条件付き確率の比で実施することとします.

$$\frac{P(M_1|y, x)}{P(M_2|y, x)} = \frac{P(y|x; M_1)}{P(y|x; M_2)} \frac{P(M_1)}{P(M_2)}. \quad (35)$$

モデルの事前確率に関しては何かしらの仮定を入れる必要がありますが, シンプルにあらゆるモデルの事前確率が同じであるとして, 右辺第一項の evidence をより詳しく見ます. まず, これまでの結果から以下のように書けます.

$$P(y|x; M) = \int d\omega P(y|\omega, x; M) P(\omega; M), \quad (36)$$

$$= \left(\frac{\lambda}{2\pi}\right)^{\frac{1}{2}} \int d\omega e^{-C(\omega; M)}. \quad (37)$$

解の性質を調べるために, モデルによる予測が ω_0 という解の寄与が支配的であるとしましょう. パラメタ ω を解 ω_0 の周りで二次まで Taylor 展開すると

$$C(\omega; M) \simeq C(\omega_0) + C''(\omega_0) \frac{(\omega - \omega_0)^2}{2}, \quad (38)$$

となります. $C(\omega_0)$ は解なので一階微分の項はゼロとなっていることに注意して下さい. これを用いて式 (37) を評価します.

$$P(y|x; M) \simeq e^{-C(\omega_0)} \left(\frac{\lambda}{2\pi}\right)^{\frac{1}{2}} \int d\omega e^{-C''(\omega_0) \frac{(\omega - \omega_0)^2}{2}}, \quad (39)$$

$$= \exp \left\{ -C(\omega_0) - \frac{1}{2} \ln(C''(\omega_0)/\lambda) \right\}. \quad (40)$$

ここで, ガウス積分 $\int_{-\infty}^{\infty} dx e^{-a(x-b)^2} = \sqrt{\frac{\pi}{a}}$ を使っています. したがって, 解における損失関数と $\log(\text{曲率}/\text{正則化係数})$ で特徴付けられることが分かります. この形から, パラメタを p 次元に拡張した場合に次の形になることは容易に想像できます.

$$P(y|x; M) \simeq \exp \left\{ -C(\omega_0) - \frac{1}{2} \sum_i^p \ln(\lambda_i/\lambda) \right\}. \quad (41)$$

ここで, λ_i は固有値です. あるモデルがどの程度良いモデルかを null model と比較することにします. null model とは $P(y|x; \text{null}) = \exp\{-N \ln k\}$ となるもので, k はクラス数を表します. $E(\omega_0) = C(\omega_0) + (1/2) \sum_i \ln(\lambda_i/\lambda) - N \ln k$ を導入することで以下の関係式を得ます.

$$\frac{P(y|x; M)}{P(y|x; \text{null})} = e^{-E(\omega_0)}. \quad (42)$$

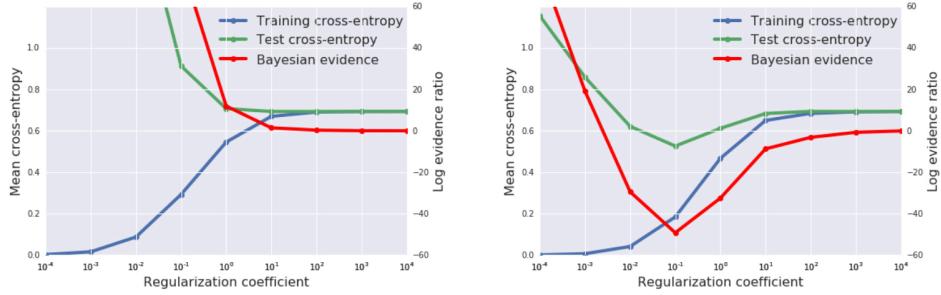


図 10 式 (42) の $E(\omega_0)$ と λ のプロット. 図は [27] より抜粋. logistic 回帰で MNIST の 0,1 を判別するタスク. 左はランダムなラベルで学習した場合で右は正しいラベルで学習した場合.

この結果はモデルの parametrization に依らず flat な解 (λ_i が小さい) が sharp な解よりも良く汎化された解であるという経験的事実を支持しています.

図 10 から意味のある情報を持つラベルでは $E(\omega_0)$ が 0 を下回ることが分かります. これはモデルが単に答えを暗記しているわけではなく意味のある解である場合は flat な解となることを示唆しています.

続いて汎化ギャップに関する実験結果を見てみましょう. 図 11 から見て取れるように、バッチサイズによって汎化性能に差が生じています. これを汎化ギャップと呼び、汎化性能をどう理解すべきか頭を悩ませるものであると同時に、実用面でも学習を早くしようと思っても迂闊にバッチサイズを大きくできないという問題を孕んでいます.

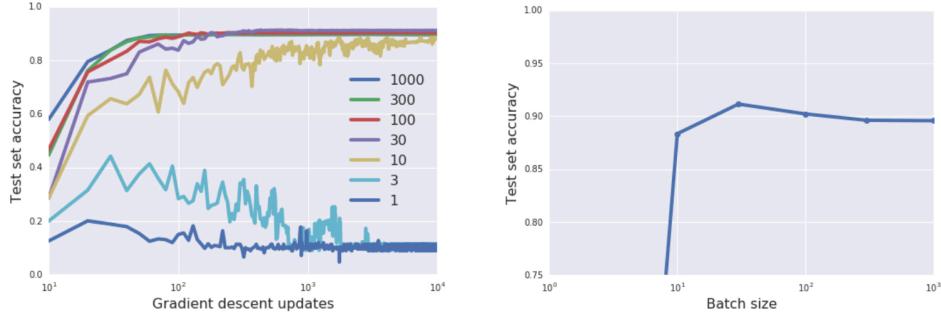


図 11 汎化ギャップ. 図は [27] より抜粋. 800 hidden units + ReLU のモデルによる MNIST 判別タスク.

以降は Stochastic Gradient Descent (SGD) に話を限定します. 汎化ギャップにおいてバッチサイズ（とデータ数の差）が重要であることに注意し、勾配によるパラメタ更新の差分を以下の形に書きます.

$$\begin{aligned} \Delta\omega &= -\frac{\epsilon}{B} \frac{d\Sigma_i^B C_i}{d\omega} = -\frac{\epsilon}{N} \frac{d\hat{C}}{d\omega}, \\ &= -\frac{\epsilon}{N} \left(\frac{dC}{d\omega} + \frac{d\hat{C}}{d\omega} - \frac{dC}{d\omega} \right) := -\frac{\epsilon}{N} \left(\frac{dC}{d\omega} + \alpha \right). \end{aligned} \quad (43)$$

ここで、 $\frac{dC}{d\omega} = \sum_i^N \frac{dC_i}{d\omega}$, $\frac{d\hat{C}}{d\omega} = \frac{N}{B} \sum_i^B \frac{dC_i}{d\omega}$ を用いました. 期待値は以下のように書けます.

$$\left\langle \frac{dC_i}{d\omega} \right\rangle = \frac{1}{N} \frac{dC}{d\omega}. \quad (44)$$

$$\left\langle \frac{dC_i}{d\omega} \frac{dC_j}{d\omega} \right\rangle = \left(\frac{1}{N} \frac{dC}{d\omega} \right)^2 + F(\omega) \delta_{ij}. \quad (45)$$

これを使うと式 (43) における α の期待値は $\langle \alpha \rangle = 0$, $\langle \alpha^2 \rangle = N \left(\frac{N}{B} - 1 \right) F(\omega) \simeq \frac{N^2}{B} F(\omega)$ と書けます ($N \gg B$ としていますが、これは deep learning の典型的な学習では成り立つ関係です).

これを確率微分方程式と比較しましょう. ここからは Langevin 方程式を解析していきますが、Appendix A.4 に簡単な説明を付したので興味がある人は読んでみてください. overdamped Langevin 方程式は以下の

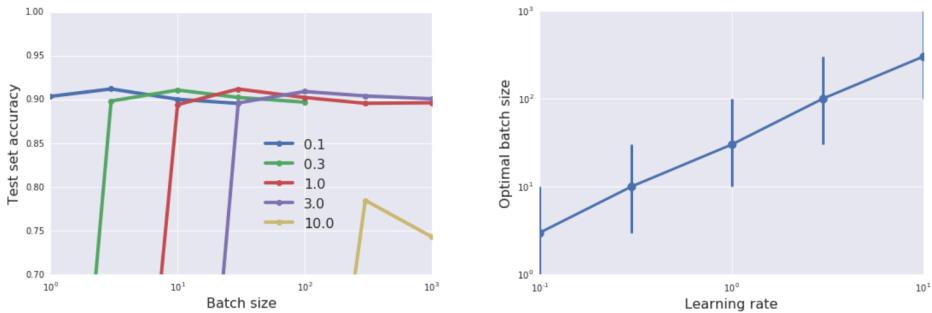


図 12 ノイズスケールと汎化性能の関係. 図は [27] より引用. 左は各線が各学習率に対応しています. 右は汎化性能を高めるバッチサイズと学習率の関係を示しています.

形のものです（付録の説明では簡単のため省いているポテンシャル項をここでは入れています）.

$$\frac{d\omega}{dt} = -\frac{dC}{d\omega} + \eta(t). \quad (46)$$

ここで, η はノイズで $\langle \eta \rangle = 0, \langle \eta(t)\eta(t') \rangle = gF(\omega)\delta(t-t')$ を満たすものとします. g はダイナミクスの揺らぎを規定する量です.

SGD における離散的なパラメタ更新の式の連続極限を取ることでこの表式と対応付けましょう. 具体的には比 $\frac{\epsilon}{N}$ が十分小さいとして以下の関係をつけます.

$$\begin{aligned} -\frac{\epsilon}{N} \left(\frac{dC}{d\omega} + \alpha \right) &= \Delta\omega = \int_0^{\epsilon/N} \frac{d\omega}{dt} dt, \\ &= -\frac{\epsilon}{N} \frac{dC}{d\omega} + \int_0^{\epsilon/N} \eta(t) dt. \end{aligned} \quad (47)$$

両辺を二乗して期待値を取ることで以下の関係式を得ます.

$$\begin{aligned} \frac{\epsilon^2}{N} \left(\frac{N}{B} - 1 \right) F(\omega) &= \frac{\epsilon}{N} gF(\omega). \\ \rightarrow g &= \epsilon \left(\frac{N}{B} - 1 \right). \end{aligned} \quad (48)$$

ここで得られたノイズスケール $g = \epsilon(N/B - 1) \simeq \epsilon(N/B)$ こそが解の汎化性能をコントロールするものだと仮定しましょう. これは適切な大きさのノイズであれば適切な解まで導いてくれるという直感的理理解に基づくものであり, 数学的に証明されているものではないことに注意してください. これを実験的に調べたものが図 12 であり, 学習率に合わせて適切なバッチサイズを取ることで同じような汎化性能の解が得られることを示しています. これは SGD に限った限定的な解析ですが, 学習率とバッチサイズの関係に良い見通しを与え, 特に大規模分散学習などにおいて大きなバッチサイズで学習する際に学習率を上げれば良いことを示唆しています*13.

この結果は大規模学習でも成立していることが示されており [28], 重要な有用なものです.

2.5 演習

Appendix A.3 にパラメタ数がデータ数より大きい場合の線形システムの解析の話が載っています. 以下の手順で解の性質を調べて下さい (Appendix に答えが書いてあるので, 自力で解きたいという人は最初の三段落だけ読んで以降は読まないように注意してください).

1. $X\omega = y$ で X が full rank であることを用いて解 ω を求める (この解を ω_l とします).

*13 momentum を入れる場合も, 少し計算が煩雑になりますが, 慣性項を含めた Langevin 方程式を同様に解析することで類似関係式を得ることができます. momentum のパラメタを m としたときに $g \simeq \frac{\epsilon N}{B(1-m)}$ という形になりますが, 詳細は原論文をご覧ください.

2. $(\omega - \omega_l)$ と ω_l が直行することを示す.
3. ω_l が L_2 ノルムの意味で最小であることを示す.
4. 文献 [29] の該当部分を読み、この性質と汎化性能の関係をまとめる.

3 画像分析：クックパッドにおける活用事例と画像分析基礎

この章では、クックパッドにおける画像分析の活用事例と画像分析の基礎に関して解説をします。

3.1 クックパッドにおける活用事例

この節ではクックパッドにおける画像分析の活用事例について紹介します。ここでは各事例について詳しく解説することはしませんが、どのような目的でどのような画像分析を使っているのかという観点で概説します。**料理きろく**

これはユーザの携帯端末中の画像から料理画像のみを抽出してカレンダー形式で表示してくれる機能（図 13 の左側）です。日々撮影される料理画像を一覧できることで、日々の料理を視覚的に振り返ることができることに加え、この画面からレシピ投稿やつくれぽ送付もできるようになっています。この機能により、日々のお弁当の記録ができて楽しみが増したり、うまく作れたものを選択的にレシピ投稿やつくれぽ送付ができたりします。20180810 時点で、処理写真枚数が 2 億 6500 万枚以上、料理と判定された写真枚数が 3200 万枚以上、利用ユーザ数が 24 万 5000 人以上となっています。

モデルの出力としては単純な二値ですが、我々のサービスの目的にマッチするように二度のアップデートを経て現在のバージョンに至ります。苦手なカテゴリをケアするためにカテゴリの種類や数を調整したり、ユーザの画像にはアクセスできないのでテスト用の画像を社内で集めたり、画像の局所的な情報を使うためにパッチ化したモデル（図 13 の右側）を使用したり、様々な工夫をしています。モデルやバックエンドの詳細な情報に関しては、以下をご覧ください。

- [料理きろくにおける料理/非料理判別モデルの詳細](#)（クックパッド開発者ブログ）
- [料理画像判定のためのバックエンドアーキテクチャ](#)（クックパッド開発者ブログ）

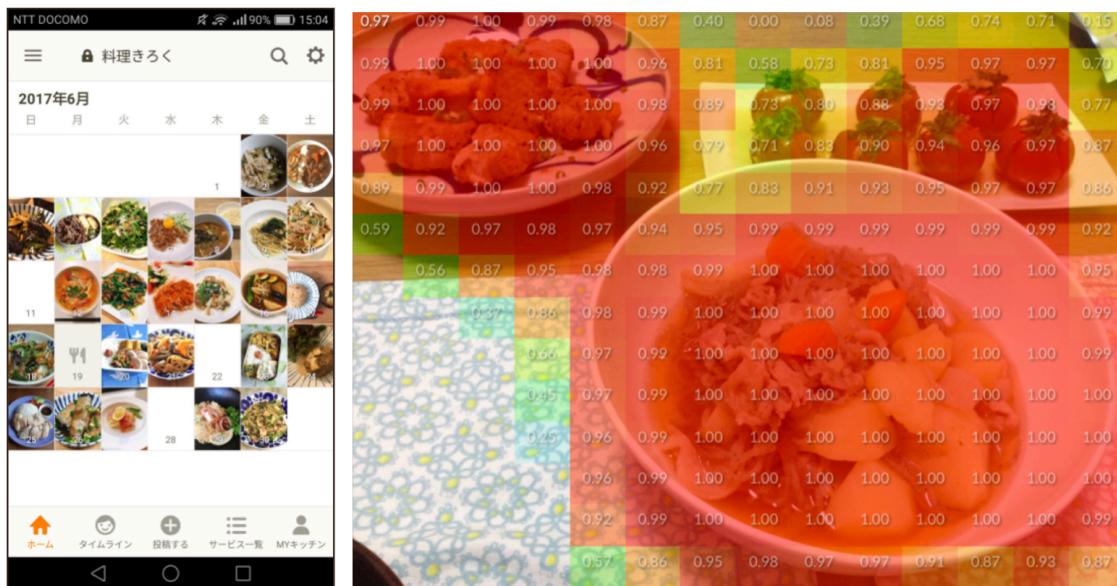


図 13 料理きろく。左の図がカレンダー形式の表示で、右の図が patch-net によるパッチ毎のスコアの結果です。

レシピ分類

これは料理きろくにおいて料理と判別された料理画像に対して、その料理画像がどのレシピなのかを分類して表示してくれる機能（図 14）です（20180810 時点では Android のみ対応）。料理/非料理からさらに先に進んで、該当の料理画像が何のレシピかを分類することで、より詳しく自分の食生活を振り返れたり、レシピのレコメンドなどに活用していくことができたりします。20180810 時点では対応カテゴリが 50 個ですが、モデル

を一新してカテゴリ数も大幅に増やしたものを作成してデプロイのために実装中です。

この問題は単純な画像の多値分類に思えますが、実際には非常に難しい問題です^{*14}。その主たる要因がカテゴリが定まっていない分類問題だからで、どのようなカテゴリを対象にして学習データを作るのかが難しく（そもそもカテゴリがうまく体系化されていない），さらに本番環境は完全に open set となっているためです。そのような状況で precision が高いモデル（サービス的にはこれが望まれます）を作るのは簡単ではなく、カテゴリ間の類似度を算出して対象カテゴリを調整したり、binary classifier を cascading させて閾値を調整したり、様々な工夫をして現在のバージョンに辿り着きました。より詳細な技術的内容に関しては、以下をご覧ください。

- ClassSim: Similarity between Classes Defined by Misclassification Ratios of Trained Classifiers (arXiv に投稿した論文)
- Solve "unsolved" image recognition problems in service applications (Cookpad TechConf 2018 での発表。この発表の中でレシピ分類も紹介しています)



図 14 料理画像のレシピ分類。

料理画像の超解像

これは低解像度の料理画像を高解像度に変換するために活用している技術です（図 15）。クックパッドは 20 年以上もサービスを継続しているため、昔の画像は比較的低解像度で最近のものと比べると荒さが目立つてし

^{*14} 実際のところ、我々も取り組む前は多値分類を解けばよいだろうという単純な認識だったのですが、実際に取り組むことで困難な点が数多くあることを理解して驚きました。

まいですが、レシピとしては素晴らしい人気も高い、というものもあります。そのような画像を高解像なものへと綺麗に変換できればユーザにとっても有用で、そのような目的で開発をしているものです。また、ユーザにレシピ本をプレゼントするという施策において、印刷時に画質が荒くなってしまうという問題に対してこの技術を適用し、実際にユーザに届く形でも利用しています。

超解像は学習データとして高解像度/低解像度の画像のペアが必要ですが、高解像度から低解像度の画像は人为的に作れるため、データ作成のコストは比較的低めです。高解像の料理画像を集め、それに人为的なノイズ(jpeg のノイズやフィルタなど)を加えることで低解像度の画像を作成し、モデルを学習しています。ただし、高解像度への変換の成否は学習データに強く依存します。具体的には、肉の画像で学習したモデルをパンに適用しても不自然なテクスチャが生成され、料理として魅力的なものではなくなってしまいます。そこで、料理画像の中でドメインを区切り、ドメイン毎にデータを収集してモデルを作成することで、望ましい超解像が実現できるように工夫しています。より詳細な内容に関しては以下をご覧ください。

- 低解像度の料理画像を超解像するための SRGAN の応用 (人工知能学会全国大会 2018 に投稿した論文)

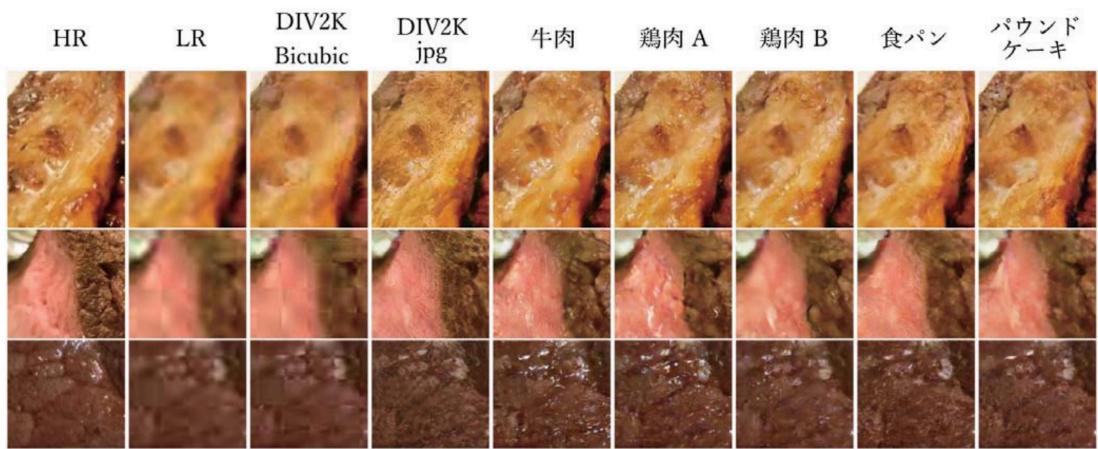


図 4: 各学習モデルによる牛肉画像の超解像結果を切り出したもの。画像は 2018 年 3 月 8 日時点で上から順に (レシピ名, レシピ作者, URL) のレシピ画像を使用。(どんな材料でもお手軽☆味噌漬け焼き, AyakoOOOOO, <https://cookpad.com/recipe/1008812>). (ローストビーフをフライパンで作ろう!, はーたんのおっかさん, <https://cookpad.com/recipe/1900893>). (安いお肉を美味しいステーキにする方法♪, ElisabethF, <https://cookpad.com/recipe/2767568>).

図 15 肉画像に対する超解像の適用例。図は [30] から引用。肉画像で学習したモデルでは適切にテクスチャを再現できていることが見て取れます。HR は元々の高解像度画像で LR は低解像度化したものを bicubic 法で拡大したものです。DIV2K は料理とは関係のない画像で、Bicubic や jpg は学習用に低解像度を作る際に使用した手法を表します。

つくれぽ画像チェック

これは日々送られてくるつくれぽの画像を自動チェックするために社内で利用している技術です。クックパッドには毎日何千件ものつくれぽが送られますが、その中には誤操作などによって対象のレシピの画像と合致していない画像が送られてくることがあります。それらを人手で全て確認するのは規模的に困難であるため、前述の料理きろくのモデルを活用して、非料理スコアが高いものを選別してそれらだけを人手で確認するという運用をしています。日次のバッチで日々のつくれぽを画像をチェックし、サポートチームの業務負担を大きく低減することができます。

料理画像の魅力度推定

料理画像の見栄えを数値化したいという目的で利用している技術です。画像が魅力的であれば料理への印象も良くなるし興味を惹きやすいため、多数ある画像の中から魅力的なものを選び出すということはサービスにとって有用です。様々な場面で料理を選定する際の基準として利用しています。モデルとしては、五段階で評点をつけた画像データを使って画像の回帰や分類を解いています。何を魅力的と思うかには個人差があるので、どんな目的においてどのようなデータセットを用いるのかということが重要なポイントとなります（例えば、

工夫して作ったお弁当は普段お弁当を作る人には凄く魅力的に見えて作らない人にとってはそうでもなかりします).

ここで紹介した以外にも、新規サービスにおける画像分析の適用可能性を探ったり、動画分析などにも取り組んでいます。画像は料理にとって重要な要素の一つなので、今後も色々な場面で画像分析を適用・活用していくたいと考えています。

3.2 画像分析の基礎

この節では画像分析の基礎に関して議論をします。ここでは具体的な数式にはあまり立ち入らず、最近の画像分析を理解する上で必要となる要素を概観するということに留めます。本当は全部を詳しく解説したいところですがそれは時間的に厳しいので、その中から convolution のみを取り上げて次節で詳しく解説することにします。

最初に、convolution を定義しておきます。一次元実関数 $f(x), g(x)$ の convolution は以下で定義されます。

$$(f \circ g)(x) := \int f(x')g(x - x') dx'. \quad (49)$$

一方で、cross correlation という関数は以下で定義されます。

$$(f \bullet g)(x) := \int f(x')g(x + x') dx'. \quad (50)$$

これらの式はかなり似ていますが、 g の中の符号の一部が flip していることに注意してください。実は、deep learning でよく使われている convolution は cross correlation のことを指しています。これはかなり混乱を招くものではありますが、慣例なので本資料でも cross correlation のことを convolution として扱っていきます。名前はあまり気にせず、次節で解説する convolution を数式レベルで理解できていればよいと思います。

最近の画像分析はほとんどが Convolutional Neural Network (CNN) に基づくものになっているので、CNN に限定して議論を展開します。CNN は名前の通り convolution を用いた neural network であり、ネオコグニトロン [31, 32] の局所結合に着想を得て、現代でも使われているような勾配法を用いて計算される形 [33] へと結実したものです。CNN の構成要素は図 16 に示されるように分解できます。これに加えて、主要なネットワークアーキテクチャについても押さえることで、CNN に関する一通りの理解を得ることができます。当然ここに載っていない新しい発展は山のようにあるのですが、それらは基本を身につければ個別に調べていけると思います。本節では、ここで登場した各要素に関して、（深入りはしませんが）何が重要なポイントなのかということを概観していきます。

convolution (畳み込み)

convolution は入力画像の特徴を抽出する学習可能なフィルタとして機能します。イメージとしては図 17 のような感じです。本来はチャンネル方向がどのように処理されているのかも考える必要がありますが、次節で詳しく調べるので、ここでは大まかな理解をするに留めます。

大きな特徴は、並進移動不变性を保って局所的に作用するようなフィルタを要請するという点です。図 17 からも見て取れるように、同じフィルタを異なる位置に適用しています。これはあるオブジェクトが異なる位置に写っていても同じものと認識するという画像データの特徴を捉えてると共に、(fully connected な場合と比べて) パラメタ数の削減にもなっています。

convolution は画像分析において本質的に重要なもので、亞種もたくさんありますが、基礎をきちんと理解することが重要なので、次節で詳しく調べます。

pooling (プーリング)

pooling は convolution で抽出した特徴マップに対して適用するもので、粗視化によってサイズを小さくすることで、微小な変化（平行移動や回転など）に対して頑健なモデルを作ることができます。粗視化には様々なパターンがありますが、図 18 に示すように、max pooling がよく使われるものです。これは入力を 2×2 を一単位とするように粗視化をして、その際に該当領域の一番大きな値を取得するような処理になっています。

Convolutional Neural Networks							
Components							
Convolutional Layer	Pooling Layer	Activation Function	Loss Function	Regularization	Optimization	Fast Processing	Applications
Tiled Convolution	Lp Pooling	ReLU	Hinge Loss	Lp-norm	Data Augmentation	FFT	Image Classification
Transposed Convolution	Mixed Pooling	LReLU	Sofmax Loss	Dropout	Weight Initialization	Structured Transforms	Object Detection
Dilated Convolution	Stochastic Pooling	PReLU	Contrastive Loss	DropConnect	SGD	Low Precession	Object Tracking
Network in Network	Spectral Pooling	RReLU	Triplet Loss		Batch Normalization	Weight Compression	Pose Estimation
Inception Module	Spatial Pyramid Pooling	ELU			Shortcut Connections	Sparse Convolution	Text Detection & Recognition
	Multi-scale Orderless Pooling	Maxout					Visual Saliency Detection
		Probout					Action Recognition
							Scene Labeling
							Speech & Natural Language Processing

Figure 1: Hierarchically-structured taxonomy of this survey

図 16 CNN の構成要素. 図は [34] から引用. これに加えて主要なネットワークアーキテクチャを理解することで CNN に関する一通りの理解を得ることができます.

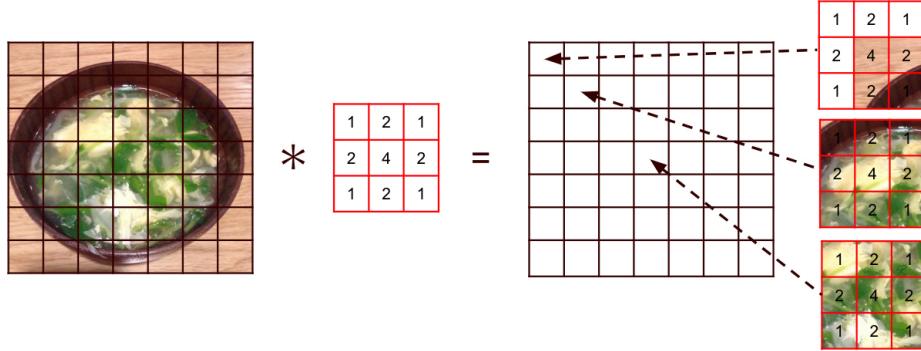


図 17 convolution のイメージ図. 入力画像に対して 3×3 のフィルタを適用しています. フィルタの値は back propagation で学習するパラメタになります. 変換後のピクセル値は元の画素のピクセル値とフィルタの要素積の後に和を取ったものになります. ここでは簡単のためチャンネル方向の情報は無視しています.

pooling は多くの場合学習パラメタのない処理になっています. また, この処理でモデルに非線形性をもたらす役割も担っています.

pooling の特殊な場合として, 変換後の height と width を 1 にする pooling のことを Global Average Pooling (GAP) と呼びます. 式で書くと $x_{i,j,c} \rightarrow x_{1,1,c} = \frac{1}{N_H N_W} \sum_{i,j} x_{i,j,c}$ (N_H, N_W は height と width のサイズ) とするような処理です. これは特徴マップをチャンネル毎の情報に集約する際に用いられる処理で, 最近の CNN の理解には欠かせない要素となっています.

activation function (活性化関数)

activation function は特徴マップに適用することでモデルに非線形性を持たせて表現力を高めるのが主な役割です. 伝統的には数学的にも取り扱いやすい sigmoid 関数 (や tanh) が使われてきました. 現在では, 様々な activation function が提案されてますが, 最も重要なものは ReLU です.

$$\text{ReLU}(x) = \max(0, x). \quad (51)$$

これはシンプルですが, 勾配消失問題に強く, 最近の CNN には必須の要素なっています. Leaky ReLU や Parametric ReLU などは ReLU の拡張になっています.

activation function は単純な関数でありながら学習に強く効く場合も多いので, 面白い研究対象でもあります. activation function の比較に関しては, 例えは [35] などを参照してください.

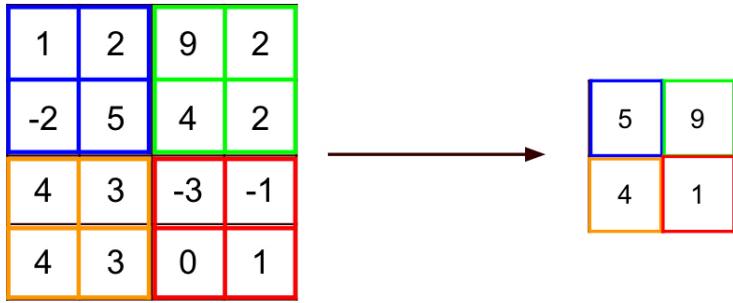


図 18 pooling のイメージ図. ここでは 2×2 で stride が 2 の max pooling を示しています.

loss function (損失関数)

loss function は back propagation によるモデルの学習の根幹を成すもので、微分可能なものを設定して、この値を小さくするようにパラメタをアップデートしていきます。典型的に使われるものはそれほど多くはありません。分類問題であれば、cross entropy がよく使われます。

$$H_y(f(x)) := - \sum_i^N y_i \log(f(x_i)), \quad (52)$$

ただし、 y が真のラベルで $f(x)$ が入力 x に対する予測としています。回帰であれば L_1 や L_2 loss がよく使われます。特微量学習によく使われるものとして triplet loss が挙げられます。これは対象とするデータ (anchor) x_i^a に対して、正のペア x_i^p と負のペア x_i^n を準備し、以下のように定めます。

$$\sum_i^N (|f(x_i^a) - f(x_i^p)|^2 - |f(x_i^a) - f(x_i^n)|^2 + \alpha), \quad (53)$$

ただし、 α はマージンです。これは特徴マップ上で似ているものをより近く、似ていないものをより遠くに配置するようにモデルを学習することになります。

loss function も新しいものが様々提案されていますので、気になる人は色々調べてみてください。

regularization (正則化)

regularization には weight decay のように explicit に効く（loss function に正則化項として加える）ものもあれば、augmentation のように implicit に効く（explicit ではないもの）ものがあります。図 16 の分類とは少し異なりますが、regularization としてこの二つを考えます。

CNN の学習においては implicit regularization の方が重要です。その中でも batch normalization [36] と data augmentation (これは多数の手法がありますが、例えば <https://github.com/aleju/imgaug> などを参照) が特に重要です。batch normalization はミニバッチ毎に平均と分散を求めて正規化をするもので、理論的には共変量シフトなどと関係します。ここでは解説をしませんが、必須の要素なので知らない人は勉強してください。data augmentation は簡単な手法で性能を大きく向上させることができるために実用上は非常に重要で、画像のラベルを変えない範囲の処理（人間が見ても対象を認識できるようなもの）を施してモデルの判別を我々が望むものに近づけます。最近では mixup [37] のように複数の画像とラベルを混合させて使うような手法も登場したりしています。

optimization (最適化)

optimization で重要なのは勾配法で学習する際にどのアルゴリズムを採用するかです。SGD や RMSProp や Adam などが有名ですが、それぞれの違いが知りたい場合は網羅的に解説した論文 [38] などを参考してください。どれを使うのが良いかは意見が分かれるかもしれません、研究としては汎化性能のためには SGD を使うのが良いと主張しているものが多い印象で、実務で使う際にはとりあえず RMSProp を使っておけば良い [39] (もしくは Adam でも可) とするものが多い気がします。

fast processing (高速化のための処理)

高速化のための各手法です。特に重要なものとして低精度の数値を使った計算や重みの圧縮など

がありますが、これらはモデルの軽量化とも深く関係しているものが多いです。これらは CNN そのものの理解というよりは少しアドバンスな内容なので、本資料では詳しくは扱いません。例えば <https://github.com/memoiry/Awesome-model-compression-and-acceleration>などを参照してください。

applications (応用例)

CNN の応用例は幅広くあります。教師有り学習ではラベルの細かさによって、画像分類、物体検出、semantic segmentation のように分けられます^{*15}。教師無し学習も盛んに調べられていて、何と言っても Generative Adversarial Networks (GAN) [40] による画像生成（や多岐にわたる応用）が重要です。

最近の応用事例を知るには [Conference on Computer Vision and Pattern Recognition 2018 \(CVPR2018\)](#)などのカンファレンスの論文を調べるのが良いです。論文の数が多くて調べるのが大変ですが、[CVPR 2018 完全読破チャレンジ](#)のように全論文のサマリを提供してくれている素晴らしいページもありますので、是非チェックしてみてください。

network architecture (ネットワークアーキテクチャ)

これまで述べてきた要素をどのように組み合わせて一つのモデルを構築するか、というのがネットワークアーキテクチャです。ここでは CNN のネットワークアーキテクチャの発展に関して特に重要なものをいくつかピックアップして、そのポイントを簡単に述べます。

- AlexNet [19]

ILSVRC 2012 で優勝したモデルで、現在の CNN の源流となっているものです。重要な要素として、ReLU, max pooling, Dropout, GPU での計算、を含んでいます。一つひとつは全く新しいというものではありませんが、それらを組み合わせて高い性能を発揮するモデルを構築したという点が重要です。特にこのモデル以降は ReLU と GPU による計算は欠かせない構成要素となりました。

- VGG [41]

これは Oxford の Visual Geometry Group (VGG) が作成したモデルで、CNN の基本的な構成要素から成るシンプルながら高い性能を誇るモデルです。現在ではモデル単独での性能は最先端のものには劣りますが、最近でも特徴量を抽出する部分に使われたりもしています。特に重要なのは 3×3 の convolution を複数層繰り返すという構造で、このモデル以降は似た構造を重ねて深い構造を形成していくということが主流になりました。

- InceptionV3 [29]

Inception は様々な種類がありますが、ここでは V3 のみを取り上げます。このモデルが登場する前後で、特徴量を異なる convolution で並列に計算してそれを結合するという構造が盛んに採用されるようになりました。Inception module (図 19) は四並列になっており、同じ base からそれぞれが異なる特徴量を学習することで高い表現力を獲得することを可能としています。Inception module のように、ネットワークアーキテクチャに特徴量の並列化を含めるという工夫はその後多くのモデルで採用されています。また、module を一つの単位としその繰り返し構造でモデルを構築していく、というのもよく使われる考え方になっています。

- ResNet [42]

図 20 のように、ショートカット結合を含めることで、勾配消失問題に対応し、適切に学習可能な深い構造を可能とするようにしたモデルです。これは、層を重ねた path で信号が消失してもショートカット結合の方の信号が生き残るために層が深くなても信号が伝播するようになっています。後述しますが、 1×1 convolution でチャンネル数を削減して計算量を削減していることもポイントです。モデルとして優秀なだけに留まらず、loss surface が滑らかになるという研究 [43] などもあり、様々な面からよく調べられているモデルになっています。

- DenseNet [44]

ResNet をさらに発展させたような構造で、図 21 のように複数ショートカット結合が張られるモデルで

^{*15} ただし最近では弱教師有り学習などもあるので、必ずしもラベルが細く付与されていなくても問題を解くことができます。

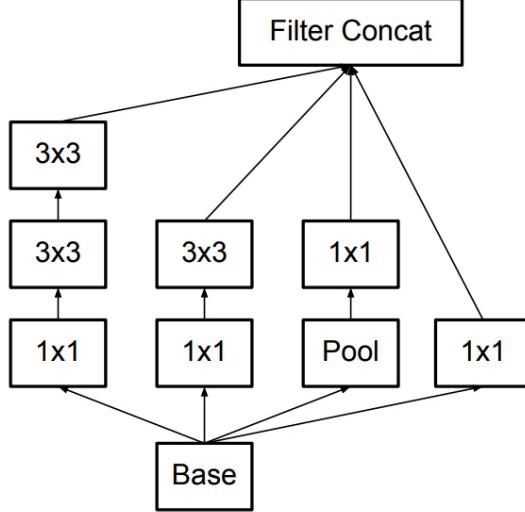


図 19 Inception module の図. 図は [29] から引用.

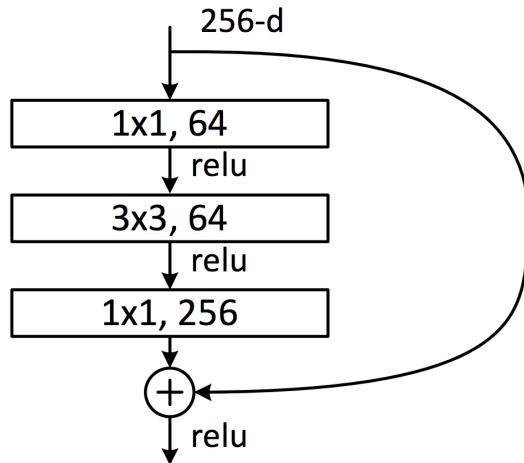


図 20 Residual block の図. 図は [42] から引用.

す. これは CVPR 2017 の Best Paper にも選ばれたもので、高い性能を発揮します. この辺りになるとネットワークアーキテクチャのトポロジーも結構複雑になってきますが、それまでに登場した主要なモデルを理解していればそれらの発展形であるということが理解できます.

- SENet [45]

チャンネル方向の情報をより活用しようとして構築されたモデルです. 図 22 にあるように、チャンネル方向の情報を GAP と非線形変換で抽出し、それを重みとして元々の特徴マップに掛け合わせることで大域的な情報も考慮できるようなモデルになっています. 20180810 時点では、手動で構築したモデルの中では ILSVRC のデータに対して SOTA なモデルになっています. チャンネル方向の情報をうまく扱う、ということは最近のモデルにおける重要な特性の一つになっています.

3.3 convolution の理解

この節では、CNN の構成要素の中でも特に重要な convolution を詳しく議論します. まず、基本的な表記を整理します. $x_{i,j,c}$ を入力データや特徴マップを含むデータを表すものとして、 i が height で j が width で c がチャンネルの index とします. また、ここでは正方形のデータのみを対象として、 i, j の範囲は同一とします.

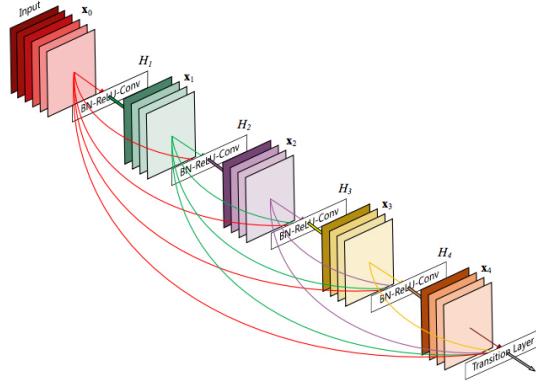


Figure 1: A 5-layer dense block with a growth rate of $k = 4$.
Each layer takes all preceding feature-maps as input.

図 21 dense block の図. 図は [44] から引用.

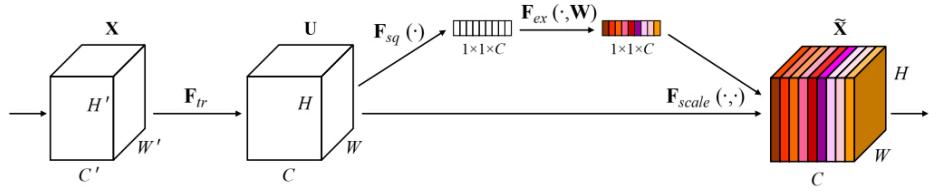


Figure 1: A Squeeze-and-Excitation block.

図 22 Squeeze-and-Excitation block の図. 図は [45] から引用.

さらに, 何層目かを表す index として l を用いて, $x_{i,j,c}^{(l)}$ のように括弧でくくることで他の index と区別するようになります.

convolution を定義します. まずは一般的な表式として次式でバイアス付きの convolution を定義します.

$$x_{i,j,c}^{(l+1)} = \sum_{i'=0}^{k-1} \sum_{j'=0}^{k-1} \sum_{c^{(l)}=0}^{C^{(l)}-1} x_{si+i',sj+j',c^{(l)}}^{(l)} w_{i',j',c^{(l)},c^{(l+1)}} + b_{i,j,c^{(l+1)}}, \quad (54)$$

ただし, i, j の index は padding で埋めた後で $\{0, 1, \dots, \frac{(N-1)+2p-(k-1)}{s}\}$ の範囲を取ります. また, $C^{(l)}$ は l 層目のチャンネル数です. 学習可能なパラメタ数は w の分が $k^2 \times C^{(l)} \times C^{(l+1)}$ で b の分が $\left(\frac{(N-1)+2p-(k-1)}{s}\right)^2 \times C^{(l+1)}$ となります. この一般的な表式は少し複雑なので, 図 23 と対応付けた上で, 簡単な例から考えていくことにしましょう. バイアス b は難しいところがないため, 以降ではゼロとして w だけを考えます.

- 入力の height と width のサイズは $N \times N$ です.
- kernel size は k とします.
- padding は p セル分だけ外埋めしている場合を考えます.
- stride は s とします.
- i, j の index は $\{0, 1, \dots, \frac{(N-1)+2p-(k-1)}{s}\}$ の範囲を取ります.

まずは最も簡単な場合から考えていきます. バイアスは難しいところがないので以降はゼロとして, まず, $k = 1, p = 0, s = 1$ という場合を考えます.

$$x_{i,j,c}^{(l+1)} = \sum_{c^{(l)}=0}^{C^{(l)}-1} x_{i,j,c}^{(l)} w_{0,0,c^{(l)},c^{(l+1)}}. \quad (55)$$

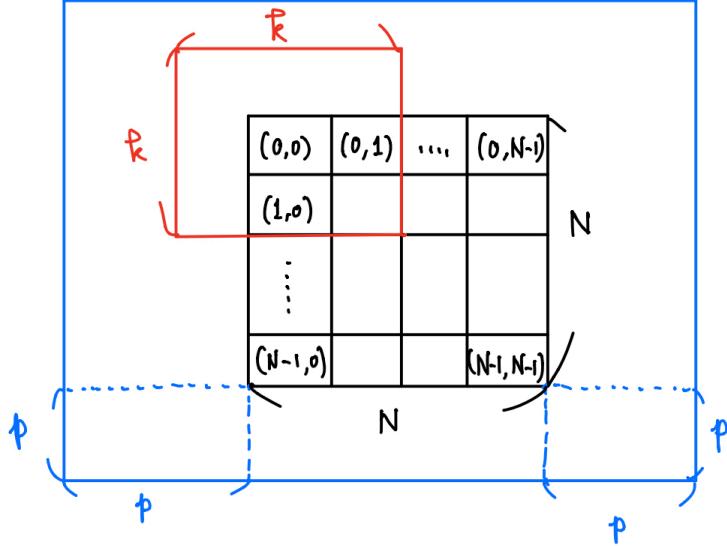


図 23 convolution の概念図. チャンネル方向は無視しています.

これは入出力の height と width の位置は一対一対応していますがチャンネル数だけを変更するような変換になっています, いわゆる 1×1 convolution になっています. i, j の index が取る値も $\{0, 1, \dots, N - 1\}$ となり整合的であることが確認できます.

次に, $k = k, p = 0, s = 1$ という場合を考えます.

$$x_{i,j,c^{(l+1)}}^{(l+1)} = \sum_{i'=0}^{k-1} \sum_{j'=0}^{k-1} \sum_{c^{(l)}=0}^{C^{(l)}-1} x_{i+i',j+j',c^{(l)}}^{(l)} w_{i',j',c^{(l)},c^{(l+1)}}. \quad (56)$$

これは kernel size k の convolution で i, j の index が取る値が $\{0, 1, \dots, (N - 1) - (k - 1)\}$ となること以外は難しいところはないと思います. kernel size が 1 よりも大きいために入力のサイズをはみださないように適切に調整されていることが理解できます.

次に, $k = k, p = p, s = 1$ という場合を考えます.

$$x_{i,j,c^{(l+1)}}^{(l+1)} = \sum_{i'=0}^{k-1} \sum_{j'=0}^{k-1} \sum_{c^{(l)}=0}^{C^{(l)}-1} x_{i+i',j+j',c^{(l)}}^{(l)} w_{i',j',c^{(l)},c^{(l+1)}}. \quad (57)$$

外側が padding されているため, 入力の height と width のサイズが $(N + 2p) \times (N + 2p)$ になっているため index の範囲に注意が必要です. i, j の index は padding で埋めた後で左上隅を $(0, 0)$ としてスタートします. また, i, j の index が取る値は $\{0, 1, \dots, (N - 1) + 2p - (k - 1)\}$ となって終了位置を調整することになります. これくらい複雑になるとどのような表記で書くかで見かけの式も結構変わってくるので混乱したりもしますが, 人の表記で気になるところがある人はぜひ自分で一から定式化してみることをオススメします.

最後に, $k = k, p = p, s = s$ という場合を考えることで, 式 (54) に戻ります. convolution を適用する開始位置が stride の分だけ飛び飛びになるので, si, sj という積になっています. この場合は先程定義した index が取る値が非整数になり得るのでその処理をどうするのかという点が厄介ですが, 例えば TensorFlow の実装では天井関数を用いて整数値にしています. 本資料では padding などとの兼ね合いでうまく割り切れて適切に演算できるものとして扱いますので, 詳しい実装は各ライブラリなどを参照してください.

transposed convolution も同じようなセットアップで定義するとします. height と width のサイズに関して, convolution は入力よりも出力が小さくなるような変換になっていますが, transposed convolution はこの逆で入力よりも出力が大きくなるような変換を指します. convolution の変換の方向を順方向とすると transposed convolution の変換の方向は逆方向となります (transposed という名前の由来もあります). そ

れゆえに, transposed convolution の処理は順方向の propagation の逆の back propagation におけるプロセスとして得ることができます, 多くの実装ではこのようにして定義しています.

もう少し具体的に議論するため, まず convolution として $N + (k - 1) \times N + (k - 1)$ の height と width に対して $\{k=k, p=0, s=1\}$ を考えます. ここでは, チャンネル数は本質的に重要ではないので変換前後で同じチャネル数を保つ convolution を考えます. データの shape は $(N + (k - 1), N + (k - 1), c) \rightarrow (N, N, c)$ となります (もう少し正確に言うと変換後の height と width のサイズが N になるように変換前のサイズを調整しています). この逆プロセスを実現するような convolution を考えましょう. $N \times N$ の height と shape に対して $\{k=k, p=k-1, s=1\}$ を考えると, $(N, N, c) \rightarrow (N + (k - 1), N + (k - 1), c)$ という変換になります. convolution の定義に戻り, height と width の index が $\{0, 1, \dots, (N - 1) + 2p - (k - 1)\} = \{0, 1, \dots, (N - 1) + 2(k - 1) - (k - 1)\} = \{0, 1, \dots, (N - 1) + (k - 1)\}$ という範囲になることに注意してください. 対となっている関係性の把握には図 24 も参考にしてください. すなわち, transposed convolution が

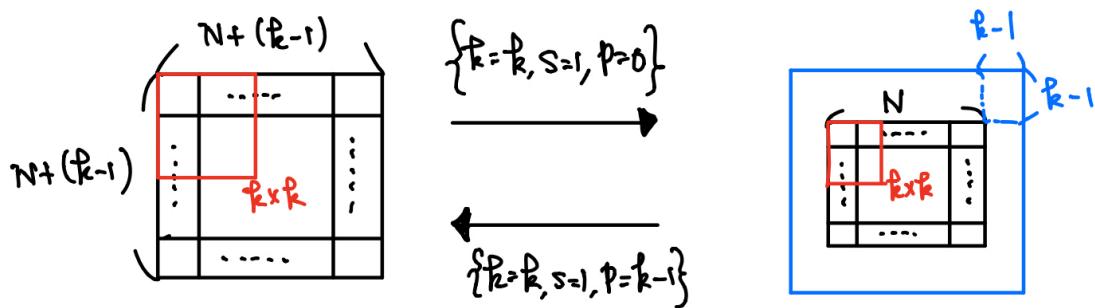


図 24 convolution (サイズが $N + (k - 1)$ で $\{k = k, p = 0, s = 1\}$) と transposed convolution (サイズが N で $\{k = k, p = k - 1, s = 1\}$) の関係性.

対応する convolution の逆プロセスで定義されることが理解できます.

続いて padding ありの場合として, height と width のサイズが $N + (k - 1) - 2p$ で $\{k=k, s=1, p=p\}$ という convolution を考えます. データの shape は $(N + (k - 1) - 2p, N + (k - 1) - 2p, c) \rightarrow (N, N, c)$ と変更されます^{*16}. これも逆プロセスを実現するような convolution を考えましょう. $N \times N$ の height と shape に対して $\{k=k, p=k-p-1, s=1\}$ を考えると, $(N, N, c) \rightarrow (N + (k - 1) - 2p, c)$ という変換になることが分かります. これも定義に戻り index の範囲を確認してみてください. 対となっている関係性の把握には図 25 も参考にしてください.

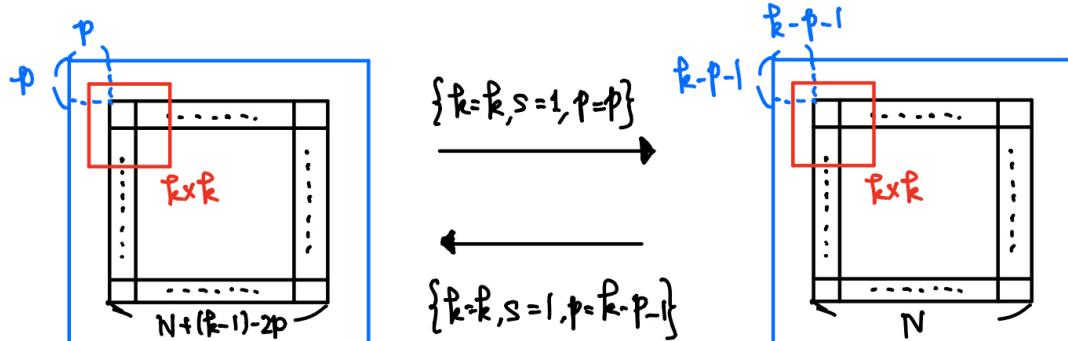


図 25 convolution (サイズが $N + (k - 1) - 2p$ で $\{k = k, p = p, s = 1\}$) と transposed convolution (サイズが N で $\{k = k, p = k - p - 1, s = 1\}$) の関係性.

最後に stride > 1 の場合として, height と width のサイズが $s(N - 1) + 1 - 2p + (k - 1)$ で $\{k=k, s=s, p=p\}$

^{*16} 入力サイズより出力サイズが小さくなる場合を考えているので $(k - 1) - 2p > 0$ です.

を考えてみます。データの shape は $s(N - 1) + 1 - 2p + (k - 1), s(N - 1) + 1 - 2p + (k - 1), c \rightarrow (N, N, c)$ と変更されます。この場合は height と width の index がどの範囲を取るかも意識しておいた方がよく、この場合は $\{0, 1, \dots, s(N - 1) - 2p + (k - 1)\}$ です。これも逆プロセスを実現するような convolution を考えましょう。 $N \times N$ の height と shape に対して $\{k=k, s=1, p=k-1-p\}$ と考えます。ただし、この場合は入力の height と width の各位置の間に $s - 1$ 個の padding を埋めることに注意してください^{*17}。この変換で $(N, N, c) \rightarrow (s(N - 1) - 2p + (k - 1) + 1, s(N - 1) - 2p + (k - 1) + 1, c)$ という変換になることが分かります。これを確認するために height と width のサイズと index の取りうる範囲を細かく見てみることにします。まず、前述の入力位置の間に入れる padding の効果を考えます。元々サイズ N であったものの間に $s - 1$ 個の padding を入れていくので、サイズは $N + (N - 1)(s - 1) = s(N - 1) + 1$ になり、index としては $\{0, 1, \dots, s(N - 1)\}$ を取ることになります。これに kernel size $k = k$ の効果と通常の外側の padding の効果 $p = k - 1 - p$ を含めるので、サイズは $s(N - 1) + 1 + 2(k - 1 - p) - (k - 1) = s(N - 1) + 1 - 2p + (k - 1)$ になります。index としては $\{0, 1, \dots, s(N - 1) - 2p + (k - 1)\}$ を取ることになります。対となっている関係性の把握には図 26 も参考にしてください。以上により、確かに transposed convolution が対応する convolution

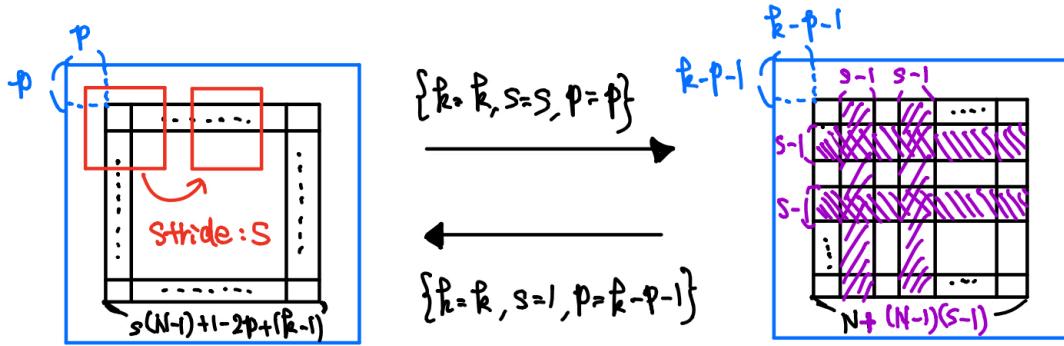


図 26 convolution (サイズが $s(N - 1) + 1 + (k - 1) - 2p$ で $\{k = k, p = p, s = s\}$) と transposed convolution (サイズが N で各位置の間に $s - 1$ の padding をして $\{k = k, p = k - p - 1, s = 1\}$) の関係性。

の逆プロセスで定義できることができました。

こういう細かい計算はないがしろにしてしまう場合も少なくないので、この機会に自分で手を動かしてちゃんと理解してみましょう。convolution には様々な亜種が存在しますが、ここで理解した基礎があれば、それらを理解することはそれほど難しくないと思われます。

3.4 演習

Sobel filter を NumPy と Pillow のみを用いて実装してください。Sobel filter が分からぬ場合は調べてください。実装に際しては、効率よりも講義で理解した convolution の数式に基づき、filter を適用するという気持ちが伝わるように意識してください。環境構築や提出場所などは GHE の repository に基づいて進めるので、講師の指示に従って進めてください。

^{*17} これは convolution の stride の逆の操作になっていることが理解できます。これはある意味で stride が 1 より小さい場合を考えることにも対応しているので、fractional stride と呼ばれたりもします。

4 画像分析：軽量モデルの理解と実データを用いた演習

CNN の中でも軽量なモデルに注目して議論をしたいと思います。軽量なモデルを作るのは以下の点で重要です。

- 使用メモリや処理時間の観点から効率的なモデルを作成
- エッジデバイスでの処理
- 比較的少ないパラメタ数で高い表現力を持つモデルの可能性の探索

軽量なモデルを作るためのアプローチには様々なものがあります。例えば、優れたネットワークアーキテクチャを構築する、低精度演算 [46]、重みの枝刈り [47]、蒸留 [48] による小さなモデルへの転換、などが挙げられます。モデルサイズの圧縮に関して一番有名な論文は [49] だと思われます。

ここでは、特にネットワークアーキテクチャに限定して、いくつかのモデルをベースにして重要な特徴を議論していくたいと思います。特に、特徴マップの空間方向とチャンネル方向の取り扱いの工夫に注目して調べていきます。

その後、実際にクックパッドで使われている画像データを使って画像分析に取り組みます。

4.1 軽量なネットワークアーキテクチャの理解

この節では、CNN の中でも軽量なネットワークアーキテクチャを売りにしているモデルをいくつか調べます。convolution のパラメタ数を思い出せば、（層間に違いがないならば）kernel size とチャンネル数が少ないほど軽量なモデルになります。典型的には kernel size は 3×3 など小さいものが使われることが多いので、チャンネル数をどのように削減するかということがポイントになります。

チャンネル数の取り扱いの工夫の一つとして ResNet の構造（図 20）を振り返ってみます。height と width のサイズは各ステップで不变であるとすると、Residual block の convolution の重み w のパラメタ数は

$$(256 \times 64) + (3 \times 3 \times 64 \times 64) + (64 \times 256) = 69632, \quad (58)$$

と計算されます。もし各ステップでチャンネル数が 256 のまま変化しないならパラメタ数は 720896 と 10 倍程度も増えてしまいます。明白ですが、ここで大きな働きをしているのは 1×1 convolution であることが分かります。ショートカット結合や並列な構造を持たせるときに、 1×1 convolution によってチャンネル数を減らしている（けれどもショートカット結合や並列構造により高い表現力を保つ）ということが重要です。

- Network In Network (NIN)[50]

NIN の大きな特徴は multilayer perceptron (MLP) convolution と GAP です。前者は (convolution が適しているという) モデルの前提を排して普遍性定理に基づき表現力の高い MLP を使うというもので、軽量なモデルとは直接は関係ないのでここでは説明を省きます。後者の GAP が重要で、これは既出ですが平均を取ることで $x_{i,j,c} \rightarrow x_{1,1,c} = \frac{1}{N_H N_W} \sum_{i,j} x_{i,j,c}$ (N_H, N_W は height と width のサイズ) とする変換です。単なる平均なので、ここには学習すべきパラメタは存在しません。チャンネル数 c を予測したいカテゴリ数と合わせることで分類問題を解くことができます。NIN ではこれを overfit を防ぐための機構として導入しています。fully connected で分類問題を解く場合、一次元の特徴ベクトル（サイズ d ）と一次元のカテゴリ数と同じサイズのベクトル（サイズ c ）をつなぐので、そのパラメタ数は典型的には $\mathcal{O}(10^3 \times 10^3)$ にもなります。これが overfit の温床になるという考えですが、それを回避するということはパラメタ数の削減そのものなので、GAP は軽量なモデルを作ると時の重要な手法となっています。また、SENet のように層の途中で空間方向の情報を集約してチャンネル毎の特徴としてスカラ値を割り当てるなど、fully connected の置き換えだけでない幅広い応用があります。

- SqueezeNet[51]

軽量なモデルの一つとして SqueezeNet も有名です。ネットワークアーキテクチャのみならず、重みの

スペース化や量子化や Huffman 符号化、低精度演算など様々な技術を複合して 1[MB] 以下のモデルを作ることに成功していますが、ここではネットワークアーキテクチャのみに注目します。fire module(図 27)と呼ばれる構成要素を定義し、これは 1×1 convolution を用いてチャンネル数を ”squeeze”しているものとなります。

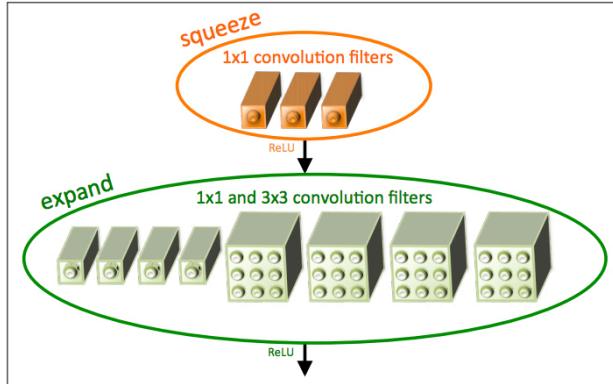


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

図 27 fire module の概念図。図は [51] から引用。この図はキューブの数がチャンネル数でキューブのサイズが kernel size に対応しています。そのため、squeeze でチャンネル数を減らした特徴マップに対して、異なる kernel size の convolution を適用してチャンネルサイズを増やしています。

- MobileNetV1[52]

MobileNetV1 は Xception [39] などでも使われた depthwise separable convolution という手法を用いて軽量なモデルを実現しています。depthwise separable convolution の特徴を捉るために、普通の convolution との違いに注目しましょう。普通の convolution のパラメタ数は $k^2 \times C^{(l)} \times C^{(l+1)}$ であって、これは kernel が作用する空間方向とチャンネル方向を同時に変換しているので積で効いています。これを避けるため、入力の $C^{(l)}$ と同じ数だけ $l+1$ 層目のチャンネルを準備し、しかもチャンネル毎に別々に計算する場合を考えてみます。

$$x_{i,j,c}^{(l+1)} = \sum_{i'} \sum_{j'} x_{i+i',j+j',c} w_{i',j',c}, \quad (59)$$

ここで、式 (49) と比べてチャンネル方向の和がなくなっていることに注意してください。チャンネル毎の計算なので、これは depthwise convolution と呼ばれ、パラメタ数は $k^2 \times C^{(l)}$ となることが分かります。パラメタ数は削減されますが、このままだとチャンネル方向の mixing が生じずに十分な表現力を獲得できないことが懸念となります。そこで、depthwise convolution の後に、 1×1 convolution でチャンネル方向を混ぜてチャンネル数も $C^{(l)} \rightarrow C^{(l+1)}$ と変化させることを考えます。 1×1 convolution のパラメタ数は $C^{(l)} \times C^{(l+1)}$ なので、合計のパラメタ数は以下のように計算されます。

$$k^2 \times C^{(l)} + C^{(l)} \times C^{(l+1)}. \quad (60)$$

これは普通の convolution のパラメタ数と比較すると、以下のように削減できていることが理解できます。

$$\frac{k^2 \times C^{(l)} + C^{(l)} \times C^{(l+1)}}{k^2 \times C^{(l)} \times C^{(l+1)}} = \frac{1}{C^{(l+1)}} + \frac{1}{k^2}. \quad (61)$$

MobileNetV1 では、これを用いて効率的なモデルを作成しています。ネットワークアーキテクチャ全体は図 28 のようになります。

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

図 28 MobileNetV1 のネットワークアーキテクチャ. 図は [52] から引用. Conv dw が depthwise separable convolution を意味しています.

- MobileNetV2[53]

MobileNetV2 は V1 からの拡張で、より軽量で高性能なモデルを実現しています。特に ReLU の持つ表現力に特化してモデリングをしている点が特徴的です。これまで ReLU は活性化関数として優秀な性能を示すことが多いので様々なモデルで使っているものでしたが、このモデルでは ReLU に特化して ReLU とうまくマッチするようなネットワークアーキテクチャを考案しています。簡単に説明すると、
 $\rightarrow \text{ReLU}(Bx)$ でゼロにならない領域は Bx で単なる線形変換

\rightarrow 一方でゼロになる領域があるために情報が落ちる部分は、チャンネル数を拡大して ReLU を適用して後で戻することでカバーできることが示せる

\rightarrow 線型活性化の後に、大きめのチャンネル数にして depthwise separable convolution と ReLU を適用して、元のチャンネル数と線型活性化に戻す

という Inverted residuals and linear bottlenecks という構造（図 29）を考案しました。もう少し詳しい内容は 第 46 回 コンピュータビジョン勉強会@関東（前編）での菊田の発表資料 や原論文を参照してください。ここで、元々の residual block は中間のチャンネル数を小さくすることでパラメタ数を削減していたことを思い出しましょう。このモデルでは中間のチャンネル数を大きくしているのでその意味でパラメタ数を削減することはできていません。そのため、例えば MobileNetV1 と比べて軽量なモデルが実現できるか否かは表現力を保ったままどれくらいチャンネル数などを減らせるかに依ります。結果としては、チャンネル数を抑えることができるので MobileNetV1 と比べて軽量で高性能なモデルを構築することができます。ネットワークアーキテクチャ全体は図 30 のようになります。

- ShuffleNet[54]

これは ResNeXt [55] で広く知られるようになった group convolution をさらに効率的に使ってモデルリングをしたものとなります。まず、group convolution とは何かを理解しましょう。前述の通り

(a) Residual block (b) Inverted residual block

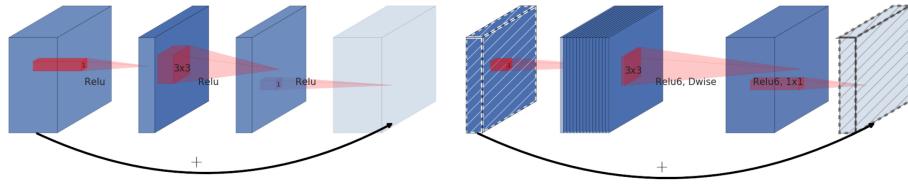


図 29 Inverted residuals and linear bottlenecks の構造. 図は [53] から引用. 各層の厚みがチャンネル数を表現し, 斜線の層は linear activation であることを意味しています.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

図 30 MobileNetV2 のネットワークアーキテクチャ. 図は [53] から引用. bottleneck が inverted residual and linear bottleneck であることを意味し, t はそれを中間の層でチャンネル数を何倍に増やすか, n は bottleneck の何回繰り返すかを表しています.

depthwise convolution は通常の convolution と比べてパラメタ数は少ないですが表現力に乏しいという問題がありました. なので, 折衷案としてチャンネルを複数のグループに分けてそれぞれのグループで通常の convolution にするというアイデアです. この group convolution は比較的良好な性能を発揮しますが, 本質的な欠点は depthwise と変わらず, グループ間では情報がやりとりされないので表現力が限定されてしまうというものです. そこで, このモデルではチャンネルをシャッフルすることでその欠点をクリアしています (図 31). ここまでチャンネル方向の取り扱いを注視してきたので, このようなモデルのモチベーションは理解しやすくなっていると思います.

- NASNet Mobile[56]

ネットワークアーキテクチャを自動的に探索するというのも一つの研究の方向性です. これまでに様々な重要な構成要素が考案されてきましたが, それらを最適に組み合わせるのは自動化した方が良いものを見つけられるだろうというアイデアです. しかし, モデル空間は広大なため, うまく探索空間を制限して探索するということがポイントとなります. この論文では, height と width を変えない normal cell と縮小をする reduction cell に分けて大枠の構造を作り, その cell の中身を 3×3 convolution や $1 \times 7 \rightarrow 7 \times 1$ convolution など典型的な部品を強化学習の枠組みで探索することで良い性能を発揮するネットワークアーキテクチャを探索しています. 計算量はかなり大きい (CIFAR-10 の実験で 2000 [GPU-hours]) ですが結果は良好です. また, 似た枠組みで計算量を落とした ENAS [57] や, 進化計算の枠組みで探索をする AmoebaNet [58], 離散最適化である探索問題を連続緩和して勾配法で最適化す

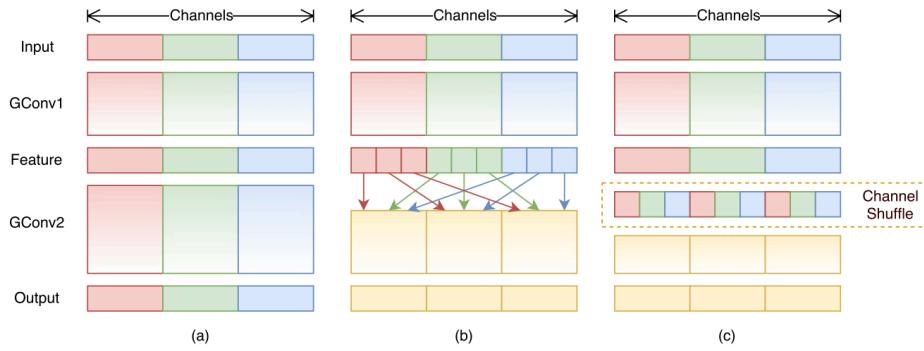


図31 チャンネルシャッフルの概念図. 図は [54] から引用.GConv が group convolution を意味しています.(a) は通常の group convolution で, (b) が GConv2 において前の層の異なるグループと結びつけていますので入力で特殊な取り扱いをしています.(c) は (b) と同等のものですが, こちらはチャンネルシャッフルを挟むことで通常の group convolution で定式化しています.

る DARTS [59] など様々なアプローチが提案されています.

ここで紹介した軽量なモデルの中でも特に最近のものである MobileNetV1 以降のモデルに関して, ImageNet の 1000 クラス分類での結果を比較しているのが図 32 です. 性能の観点からは MobileNetV2, ShuffleNet, NASNet mobile には大きな差は見られません. 現時点では, 様々なタイプの学習済みモデルが提供されているところからも, MobileNetV2 (もしくは量子化バージョンが公開されていて非常に軽量なモデルが利用できるという観点から MobileNetV1) を使用するのが良いかと思われます.

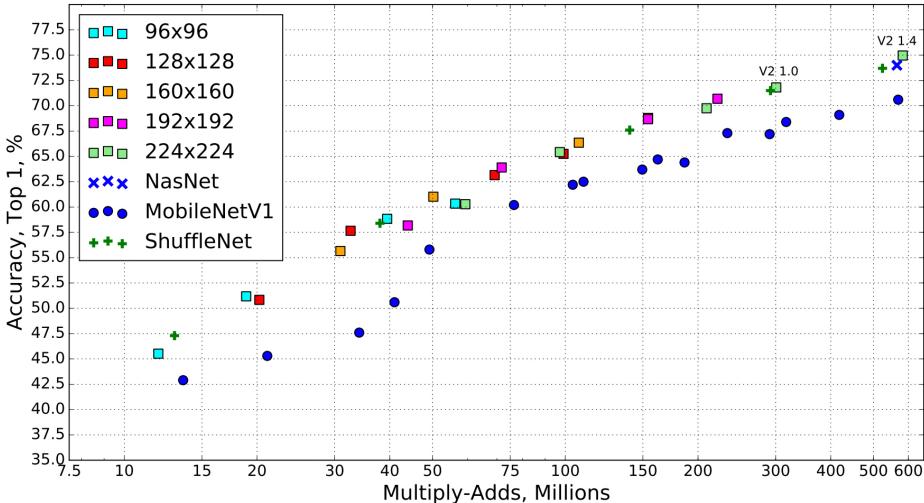


図32 各軽量モデルの ImageNet 1000 クラス分類問題における Top 1 accuracy の比較. 横軸は推測時に必要な積と和の演算数です. 図は [53] から引用. 色付きの四角の記号は全て MobileNetV2 を表しています. V2 1.0 は通常の MobileNetV2 で, V2 1.4 はチャンネル数を 1.4 倍にしてより高い性能を発揮できるようにしているモデルです.

4.2 実際の画像データを使った分析

この節では実際にクックパッドで使われている画像データを使って画像分析をしてみます. 以降は GHE の repository に基づいて進めるので, 講師の指示に従って進めてください.

付録 A Appendix

A.1 テンソル

有限次元ベクトル空間 V とその双対空間 V^* を考えます。まず、体 K 上のベクトル空間 V は、 $u, v, w \in V$ で $a, b, 1 \in K$ としたときに以下を満たすものです。

$$(u + v) + w = u + (v + w), \quad (62)$$

$$u + v = v + u, \quad (63)$$

$$\exists 0 \in V \text{ s.t. } v + 0 = v, \quad (64)$$

$$\exists -v \in V \text{ s.t. } v + (-v) = 0, \quad (65)$$

$$a(bv) = (ab)v, \quad (66)$$

$$1v = v, \quad (67)$$

$$a(u + v) = au + av, \quad (68)$$

$$(a + b)v = av + bv. \quad (69)$$

双対空間とは $f : V \rightarrow K$ という写像のうち、 $\phi, \rho \in V^*, v \in V, a \in K$ としたときに以下を満たすものです。

$$(\phi + \rho)(v) = \phi(v) + \rho(v), \quad (70)$$

$$(a\phi)(v) = a(\phi(v)). \quad (71)$$

特に、 V の基底を $\{e_i, \dots, e_n\}$ としたとき、 $e^i(e_j) = \delta_j^i$ という V^* の基底 $\{e^i, \dots, e^n\}$ を双対基底と呼びます。

ここでテンソルを定義する準備が整いました。 (p, q) -型テンソルとは、以下のような多重線型写像 T として定義されます。

$$T : V^* \otimes \cdots \otimes V^* \otimes V \otimes \cdots \otimes V \rightarrow \mathbb{R}. \quad (72)$$

ただし V^* は p 個のコピー、 V は q 個のコピーがあるものとします。 (p, q) -型テンソルを V, V^* の基底に適用することを考えます。

$$T_{j_1 \dots j_q}^{i_1 \dots i_p} := T(e^{i_1}, \dots, e^{i_p}, e_{j_1}, \dots, e_{j_q}). \quad (73)$$

これで成分を得ることができます。 $(p + q)$ 次元配列が構築できます。

簡単な場合として添字が一つの場合を考えます。ここからは成分と基底を明確に区別するために基底を太字で表示することにします。ベクトルは $\mathbf{v} = \sum_i a^i \mathbf{e}_i := a^i \mathbf{e}_i$ と書けます。和の記号を省略しているのは Einstein の縮約記法と呼ばれるものです。また、成分が上付き添字になっているのは、 $a^i = \mathbf{e}^i(\mathbf{v})$ と得られることに由来しています。これをテンソルの定義と対応させましょう。 \mathbf{e}^i が $\mathbf{e}_i \in V \rightarrow \mathbb{R}$ へと写すことを考えれば、これは $(0, 1)$ -型テンソルとなります。逆に \mathbf{e}_i は $\mathbf{e}^i \in V^* \rightarrow \mathbb{R}$ へと写す（双対の双対を考えればよいです）ことを考えれば、これは $(1, 0)$ -型テンソルとなります。

次に添字が二つのものも考えていきます。 V の基底 \mathbf{e}_i を新しい基底 $\mathbf{e}_i^{(new)}$ に変換することを考えます。

$$\mathbf{e}_j^{(new)} = A_j^i \mathbf{e}_i. \quad (74)$$

ここで、 A_j^i が少し唐突に思われるかもしれないですがこれが何者なのかを考えてみます。線形写像として $\alpha : V \rightarrow V$ を考えると、 $T(\mathbf{f}, \mathbf{v}) = \mathbf{f}(\alpha \mathbf{v})$ という $(1, 1)$ -型テンソルが構築できます。それゆえに基底を指定すれば $A_j^i = \mathbf{e}^i(\alpha \mathbf{e}_j)$ という形で成分を得ることができるというわけです。同様に双対基底の変換を $\mathbf{e}_{(new)}^i = B_j^i \mathbf{e}^j$ とすれば、

$$\delta_j^i = \mathbf{e}_{(new)}^i(\mathbf{e}_j^{(new)}) = B_j^i A_j^{i'} \mathbf{e}^{i'}(\mathbf{e}_{i'}) = B_j^i A_j^{i'} \delta_{i'}^{i'} = B_{i'}^i A_j^{i'}, \quad (75)$$

となるので、 $B_j^i = (A^{-1})_j^i$ となることが分かります。これらを用いて重要な性質であるベクトルが基底のとり方に依存しないことも示すことができます。

$$\mathbf{v} = a_{(new)}^i \mathbf{e}_i^{(new)} = (A^{-1})_j^i A_i^k a^j \mathbf{e}_k = \delta_j^k a^j \mathbf{e}_k = a^j \mathbf{e}_j. \quad (76)$$

成分と基底の変換が打ち消し合って不变性が保たれていることに注意してください。よく座標変換前後における関係性を調べたりしますが、それは成分もしくは基底のどちらかのみに注目している場合が多いです（例えば直交座標と極座標の変換）。ここで示したものは簡単な場合ですが、添字の数が増えててもテンソル積を考えることで同様に拡張していくことが可能です。

話としてはごくごく入り口ですが、テンソルに関する説明はこれくらいにしておきます。多くの人が「テンソル」と言うとき、標準的（と思われる）基底を想定してそれに適用したテンソルから得られる成分を多次元配列として扱っているものを指していることが多いと思われます。ここでの説明は簡素なものなので、興味がある人はもっとちゃんと勉強してみてください。

A.2 dual number による forward mode の定式化

dual number を以下のように Taylor 展開の一次で打ち切った場合の ϵ として定義します。

$$v + \dot{v}\epsilon. \quad (77)$$

ここで、 $v, \dot{v} \in \mathbb{R}$ です。dual number とは $\epsilon \neq 0$ で $\epsilon^2 = 0$ を満たすものです。より一般には nilpotent と呼ばれる性質であり、これは例えば物理では電子などのフェルミオンの定式化で使われる概念ですので、興味があれば調べてみてください（ただし、物理的な意味まで理解しようと思ったら結構長い道のりなので数年くらい掛かるかもしれません）。

この dual number を用いれば、関数の微分は次のように書けます。

$$f(v + \dot{v}\epsilon) = f(v) + \frac{df(v)}{dv}\dot{v}\epsilon. \quad (78)$$

これを使ってもう少し複雑な例を考えてみます。関数が入れ子になっている場合の連鎖律も期待通りに振る舞うことが示せます。

$$f(g(v + \dot{v}\epsilon)) = f(g(v)) + \frac{df(g(v))}{dg(v)} \frac{dg(v)}{dv} \dot{v}\epsilon. \quad (79)$$

これにより forward mode で用いる典型的な連鎖律が dual number の言葉で表現できることが分かります。

ここまで観察から、ある関数の微分が引数の v を $v + \epsilon$ として ϵ の係数を読み取ることで得られることが分かります。

$$\left. \frac{df(x)}{dx} \right|_{x=v} = \text{Coeff. of } \epsilon [f(v + \epsilon)]. \quad (80)$$

簡単な例で確認してみましょう。 $f(x) = e^x + 3x^2 + 4x$ の微分の値は

$$\begin{aligned} f(x) &\rightarrow f(x + \epsilon), \\ &= e^x e^\epsilon + 3(x + \epsilon)^2 + 4(x + \epsilon), \\ &= e^x (1 + \epsilon) + 3(x^2 + 2x\epsilon) + 4(x + \epsilon), \\ &= e^x + 3x^2 + 4x + \epsilon(e^x + 6x + 4), \end{aligned} \quad (81)$$

という計算から $e^x + 6x + 4$ で確かに求まっていることが分かります。

これをどのようにうまく実装するのかという点に関しては別途議論が必要ですが、ここではそこまでは踏み入れないことにします。

A.3 パラメタ数がデータ数よりも大きな線形システム

N 個のデータセット $\{(x_i, y_i)\}$ で、 x は d 次元の特徴量で y はラベルである場合に、以下の問題を解くことを考えます。

$$\min_{w \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(w^T x_i, y_i). \quad (82)$$

いま、パラメタ数がデータ数よりも大きい場合を考えるので、 $d > N$ でこの場合はどんなラベルでもフィットすることができます。 X を i 番目の行列が x_i^T である $N \times d$ の行列とすれば、 $Xw = y$ という定式化ができます。 X のランクが N 以上であれば無限個の解が存在しますが、それらの解の性質はどのように異なるのかということに興味が湧きます。

例えば、大域解近傍（全ての x_i において $w^T x_i = y_i$ となる点の近傍）の曲率で解の性質を調べると、全て同じになります。これは

$$\nabla^2 \frac{1}{N} \sum_{i=1}^N \text{loss}(w^T x_i, y_i) = \frac{1}{n} X^T \text{diag}(\beta) X, \quad \left(\beta_i := \left. \frac{\partial^2 \text{loss}(z, y_i)}{\partial z^2} \right|_{z=y_i}, \forall i \right), \quad (83)$$

から、二階微分の項は全て $w^T x_i = y_i$ という点で評価されるために Hessian が w に依らなくなり、全ての大域解で縮退するからです。非線形システムでは解の周りでの曲率に注目して議論しましたが、線形システムでは意味をなさないことが分かります。

他のアプローチとして SGD で解くことを考えてみます。ある点 (x_i, y_i) に対するパラメタの更新式は以下のように書けます。

$$w_{t+1} = w_t - \eta_t \left. \frac{\partial \text{loss}(z, y_i)}{\partial z} \right|_{z=w_t^T x_i} x_i. \quad (84)$$

初期値を $w_0 = 0$ とすれば、解が $w = \sum_{i=1}^N \alpha_i x_i$ の形で書けることが分かります。この結果と $Xw = y$ とを合わせれば、

$$XX^T \alpha = y, \quad (85)$$

という形に帰着します。 XX^T という形（いわゆるグラム行列）で d 次元方向を潰して $N \times N$ 行列となっているため、この α は unique solution を持つます。この解は CIFAR10 や MNIST で良い汎化性能を示すことが報告されています [29]。

ここで求めた解の性質をもう少し詳しく調べてみましょう。 $X\omega = y$ で X を full rank とすれば、 $\omega_l = X^T(XX^T)^{-1}y$ という解が構築可能です（代入すれば容易に確かめられます）。この解と $(\omega - \omega_l)$ が直行していることが示せます。

$$(\omega - \omega_l)^T \omega_l = (\omega - \omega_l)^T X^T(XX^T)^{-1}y \quad (86)$$

$$= (X(\omega - \omega_l))^T(XX^T)^{-1}y \quad (87)$$

$$= 0. \quad (88)$$

ここで、解を用いて元の方程式を $X(\omega - \omega_l) = 0$ という形に書き直せることを使っています。 $(\omega - \omega_l) \perp \omega_l$ であることを用いれば

$$\|\omega\|^2 = \|\omega_l + \omega - \omega_l\|^2 = \|\omega_l\|^2 + \|\omega - \omega_l\|^2 \geq \|\omega_l\|^2, \quad (89)$$

のように L_2 ノルム最小の解であることが示せます。上述のように、このノルム最小の解は実験的に良い汎化性能を発揮することが示されています。汎化性能と解のノルムの最小性が関係づいているように見えて興味深いですが、残念ながら実験的に L_2 ノルムがこれより大きくとも高い汎化性能を上回る解の存在が発見されています [29]。線形システムであっても一筋縄ではいかない、ということは問題の奥深さを感じさせるものですが、ここでは解説はこれくらいにして今後の発展を楽しみにしたいと思います。

A.4 物理学における Langevin 方程式

物理に馴染みのない読者のために Langevin 方程式を簡単に解説しておきます。質量が m でランダム力 $\eta(t)$ を受ける粒子の一次元ブラウン運動^{*18}を考えます。この粒子は花粉を水に溶かした場合の微粒子などに対

^{*18} ブラウン運動とは熱的運動によって引き起こされるランダム運動で、Einstein が出した論文 [60] により解明されたものです。

応していて、速度 $\frac{dx}{dt} = v$ においては摩擦力 $-\frac{1}{\gamma}v$ が働くものとします (γ は定数)。ここでランダム力としてはホワイトノイズを考え、デルタ関数を用いると以下のように書けます (M は定数)。

$$\langle \eta(t)\eta(t') \rangle = 2M\delta(t-t'). \quad (90)$$

この粒子の運動方程式は以下の形となり、これが Langevin 方程式と呼ばれるものです。

$$m\frac{d^2x}{dt^2} = -\frac{1}{\gamma}\frac{dx}{dt} + \eta(t). \quad (91)$$

まず、 $\left|m\frac{d^2x}{dt^2}\right| << \left|\frac{1}{\gamma}\frac{dx}{dt}\right|$ という状況を考えれば運動方程式で慣性項を無視することができ（これを overdamped Langevin 方程式と呼びます）， $x(0) = 0$ と原点を取ることで次の形式的な解を得ます。

$$x(t) = \gamma \int_0^t du \eta(u) \quad (92)$$

両辺を二乗して期待値を取ることで、次の形を得ます。

$$\langle [x(t)]^2 \rangle = \gamma^2 \int_0^t du \int_0^t du' \langle \eta(u)\eta(u') \rangle = 2M\gamma^2 t := 2Dt. \quad (93)$$

これがブラウン運動の重要な性質となります。位置座標の二乗期待値が t の一次であり進まないという意味で醉歩などとも呼ばれます。

慣性項を無視しない場合は $\frac{dx}{dt} = v$ の解として次が得られます（基本的な微分方程式の解法なので興味がある人は自分で解いてみてください）。

$$v(t) = e^{-\frac{1}{m\gamma}t} v(0) + \int_0^t du \frac{\eta(u)}{m} e^{-\frac{1}{m\gamma}(t-u)}. \quad (94)$$

両辺を二乗して期待値を取ります。cross term が 0 になることに気を付ければ ($v(0)$ と時刻 0 以降のランダム力に相関はない)，以下が得られます。

$$\langle [v(t)]^2 \rangle = e^{-\frac{2t}{m\gamma}} \langle [v(0)]^2 \rangle + \frac{M\gamma}{m} (1 - e^{-\frac{2t}{m\gamma}}). \quad (95)$$

十分に時間が経過した ($e^{-\frac{2t}{m\gamma}} << 1$) 後、エネルギー等分配則 $\frac{1}{2}m\langle [v(t)]^2 \rangle = \frac{1}{2}k_B T$ (k_B はボルツマン定数， T は温度) を用いることで、最終的に以下の関係式が導けます。

$$D = \gamma k_B T. \quad (96)$$

これはランダム力で誘起される揺動と摩擦と温度で誘起される散逸が関係づいているため揺動散逸定理と呼ばれ、Einstein の関係式とも呼ばれます。

ここでは完全に物理現象を対象に議論をしましたが、本編でも登場したように SGD を確率微分方程式として扱う場合などで機械学習の世界で遭遇するので、検索すればいくつも論文を見つけることができます。

参考文献

- [1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
- [3] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–153, 2017.
- [4] Wolfram Mathematica. Wolfram research. Inc., Champaign, Illinois, 2009.
- [5] Maplesoft. <https://www.maplesoft.com/>. Accessed: 2018-07-24.
- [6] Maxima. <http://maxima.sourceforge.net/>. Accessed: 2018-07-24.
- [7] Louis B Rall. Automatic differentiation: Techniques and applications. 1981.
- [8] Andreas Griewank et al. On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107, 1989.
- [9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.
- [10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- [11] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.
- [12] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.
- [13] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [14] Hirotugu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998.
- [15] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.
- [16] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [17] Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *arXiv preprint arXiv:1505.03654*, 2015.
- [18] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [20] Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at

- memorization in deep networks. *arXiv preprint arXiv:1706.05394*, 2017.
- [21] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.
- [22] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pages 586–594, 2016.
- [23] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.
- [24] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.
- [25] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.
- [26] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.
- [27] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. 2018.
- [28] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- [29] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [30] 永野雄大 and 菊田遥平. 低解像度の料理画像を超解像するための srgan の応用. In *人工知能学会全国大会論文集 2018 年度人工知能学会全国大会 (第 32 回) 論文集*, pages 3A103–3A103. 一般社団法人 人工知能学会, 2018.
- [31] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.
- [32] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.
- [33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [34] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, et al. Recent advances in convolutional neural networks. *arXiv preprint arXiv:1512.07108*, 2015.
- [35] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. 2018.
- [36] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [37] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- [38] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [39] Francois Chollet. *Deep learning with python*. Manning Publications Co., 2017.
- [40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,

- Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [43] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.
- [44] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.
- [45] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017.
- [46] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.
- [47] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.
- [48] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [49] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [50] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.
- [51] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [52] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [53] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- [54] Mengxiao Lin Jian Sun Xiangyu Zhang, Xinyu Zhou. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.
- [55] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.
- [56] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.
- [57] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.
- [58] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.
- [59] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv*

preprint arXiv:1806.09055, 2018.

- [60] Albert Einstein. Investigations on the theory of the brownian movement. *Ann. der Physik*, 1905.