京都大学人工知能研究会

KaiRA

Kyoto univ. AI Research Association

2022 NF

vol.6

目次

はじめに

本誌を手に取っていただきありがとうございます．KaiRA 会長の三宅です．

今となっては「AI」や「最適化」といった言葉を毎日耳にするようになりました．特に今年は DALL-E2 に始まり，Imagen や Midjourney，StableDiffusion 等，いわゆる「AI お絵描き」の分野も盛んになり、AI というものが読者の皆さんにとってさらに身近なものになったと思います．

KaiRA では「人工知能・AI」といった分野について勉強したいという京大生が集まり，毎週例会を開催しています．例会では主に教科書の輪読会を行っています．また，NF 等では実際に実装を行い作品制作も行っています．今回の NF でもいくつか作品展示をしていますので，ぜひデモを体験して頂きたいと思います．デモを通して AI の面白さに気づいて頂ければ幸いです．

KaiRA では長期休みを除いて年中見学と入会を受け付けております．入会制限もございません．現在の参加者の学部・回生も多岐に渡ります．またオンラインでの参加も可能です．もし KaiRA の活動にご興味を持たれた方は，いつでもご連絡ください！(連絡先はメール，Twitter，HP の Contact のお問い合わせフォームをご用意しています)

京都大学人工知能研究会代表
工学部 3 回生 三宅大貴

第 1 章

深層学習入門

深層学習という言葉をご存知でしょうか。近年、人工知能の発達は大きな話題を呼んでいますが、この発達には深層学習と呼ばれる技術が大きな役割を果たしています。実際に、顔認証や、音声認識、機械翻訳など、様々な場面で深層学習が使われています。

本章では、その深層学習の仕組みを紹介します。まず、深層学習のベースになっているニューラルネットワークを紹介し、次にデータから学習する方法を紹介します。この章を通して、深層学習がどのようなものなのか、わかっていただけたら幸いです。

1.1 パーセプトロン

本節では、パーセプトロンというアルゴリズムについて紹介します。パーセプトロンは、1957 年に考案されたアルゴリズムで、ニューラルネットワークの原型になっています。パーセプトロンを学ぶことで、ニューラルネットワークに関して重要な考えを学ぶことができます。

1.1.1 パーセプトロンとは

パーセプトロンは、複数の信号を入力として受け取り、一つの信号を出力します。パーセプトロンの信号は「流す or 流さない (1 か 0)」の 2 値です。パーセプトロンには、重みとバイアスというパラメータが存在します。

今から、入力信号数が 2 であるパーセプトロンの説明をします。入力信号数が増えても、同じように拡張できます。入力信号 x_1 と x_2 を受け取るとし、重みを w_1、w_2 として、バイアスを b とします。このとき、パーセプトロンの出力値 y は、

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 + b \leq 0) \\ 1 & (w_1x_1 + w_2x_2 + b > 0) \end{cases} \tag{1.1}$$

となります。これは、x_1x_2 平面に直線を引いて、その直線よりも上が $y = 1$ で下が $y = 0$、または上が $y = 0$ で下が $y = 1$ となるようにすることを表しています。パーセプトロンのモデルを図 1.1 のようによく書きます。

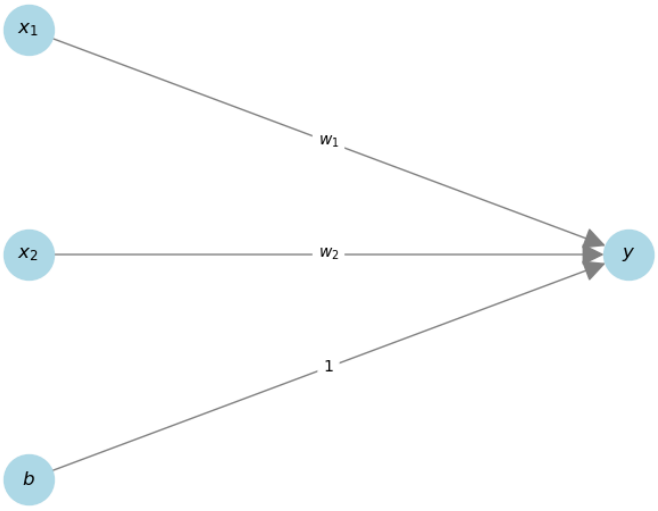

図 1.1　パーセプトロンの図

1.1.2　単純な論理回路

パーセプトロンを用いると、単純な論理回路を実装することができます。ここでは、AND ゲートと呼ばれるものについて考えてみます。AND ゲートは、入力値 x_1,x_2 を受け取り、出力値 y を出力するとすると、入力値と出力値の関係 (真理値表) は表 1.1 のようになります。

表 1.1　AND ゲートの真理値表

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

表 1.1 のように振る舞うパラメータの選び方は無限個あり、$(w_1, w_2, b) = (0.5, 0.5, -0.7)$ などとすると、うまくいきます。他にも、OR ゲート、NAND ゲートと呼ばれる論理回路も実装することができます。OR ゲートと NAND ゲートの真

理値表はそれぞれ表 1.2、表 1.3 のようになっています。ここでは、パラメータをどのように取ればよいのかについては言及はしません。良かったら考えてみてください。

表 1.2 OR ゲートの真理値表

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

表 1.3 NAND ゲートの真理値表

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

1.1.3 パーセプトロンの限界

XOR ゲートという論理回路をパーセプトロンで実現することを考えます。XOR ゲートの真理値表は表 1.4 のようになっています。

表 1.4 XOR ゲートの真理値表

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

実は、この XOR ゲートはパーセプトロンで実現することができません。x_1x_2 平面に直線を引いて、直線との上下の位置関係によって y の値を決定するというやり方では、真理値表を表 1.4 のようにできないためです。

1.1.4　多層パーセプトロン

XOR ゲートを作るためには、いくつか方法がありますが、その方法の一つを紹介します。それは、AND ゲート、NAND ゲート、OR ゲートを組み合わせて作るというものです。どのように組み合わせればよいかというと、

$$XOR(x_1, x_2) = AND(NAND(x_1, x_2), OR(x_1, x_2)) \tag{1.2}$$

というように組み合わせればよいです。このように、パーセプトロンを多層にすると、より複雑に出力を制御できます。

1.2　ニューラルネットワーク

ニューラルネットワークは、入力値 x、出力値 y に対して $y = f(x)$ となる f の近似を求めるようなモデルであり、近年盛んに使われています。特に、画像分野における発達が著しいです。ニューラルネットワークには様々な種類がありますが、本節では一番基本的な全結合層と呼ばれるニューラルネットワークを紹介します。

1.2.1　全結合層とは

全結合層とは、隣り合う二つの層のノードが全て結合しているようなニューラルネットモデルです。図 1.2 のようになっています。$x_{1,1}$,$x_{1,2}$ が入力です。1 の部分をバイアスと呼びます。以下ではこの例を使って説明をします。それぞれの層にあるノードの数を変えることも可能です。

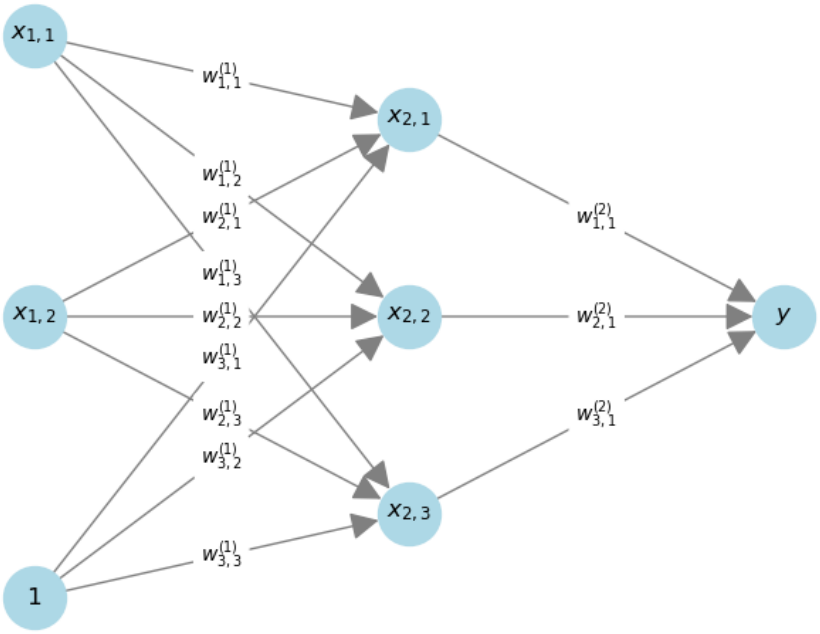

図 1.2　全結合層の図

1.2.2　活性化関数

全結合層の $x_{2,1}$ は、

$$x_{2,1} = f(w^{(1)}_{1,1}x_{1,1} + w^{(1)}_{2,1}x_{2,1} + w^{(1)}_{3,1}) \tag{1.3}$$

と計算します。他のノードに関しても同じ様に計算します。この式に現れる f というのが活性化関数です。この f は様々な関数に変えることが可能です。多層パーセプトロンに用いられていたのは、ステップ関数と呼ばれる関数です。ステップ関数は、

$$f(x) = \begin{cases} 0 & (x < 0) \\ 1 & (x \geq 0) \end{cases} \tag{1.4}$$

と表されます。活性化関数には非常に多くの種類が存在します。中間層における活性化関数としては、ReLU 関数、シグモイド関数、swish 関数などが良く用いられます。出力層における活性化関数としては、シグモイド関数、ソフトマックス関数、恒等関数、tanh 関数などが良く用いられます。それぞれの関数の形は、表 1.5 を見てください。ソフトマックス関数だけは、後で説明します。

表 1.5　活性化関数の種類

恒等	ReLU	シグモイド	tanh	swish
x	$max(x, 0)$	$\frac{1}{1+e^{-x}}$	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$\frac{x}{1+e^{-x}}$

後で扱いますが、ニューラルネットワークが進歩した理由の一つに、勾配法があります。しかし、この勾配法にも勾配消失問題という問題があり、勾配消失問題が起こりにくいことが活性化関数に求められています。当然、勾配法を用いるために、微分して良い性質が得られる関数であることも求められています。興味のある方は、例として挙げた活性化関数を微分してみるとよいでしょう。

1.2.3　出力層の活性化関数

出力層 (y を求める層) では、出力として適した形が出てくるように活性化関数を選ぶ必要があります。(後で出てくると思いますが、誤差関数が求めているような分布を出す出力層の活性化関数を選ぶのも大事です。)

出力層では、ニューラルネットワークをどのような問題に対して適用しようとしているのかが大事です。それでは、機械学習における問題の分類をしていきましょう。機械学習における問題は主に**回帰問題**と、**分類問題**に分けることができます。さらに、分類問題は、**2 値分類**と、**多クラス分類**に分けることができます。**回帰問題**とは、関数の形を予測するような問題です。**分類問題**とは、データを分類するような問題です。**2 値分類問題**はデータを 2 種類に分類する問題、**多クラス分類問題**はデータを 3 種類以上に分類する問題です。

- 回帰問題
- 分類問題
  - 2 値分類問題
  - 多クラス分類問題

図 1.3　機械学習の問題の分類

2 値分類と多クラス分類を区別する理由は、2 値分類問題には、2 値分類問題だからこそできるアプローチがあるからです。2 値分類問題を多クラス分類問題のように扱うこともできます。2 値分類問題では、出力層のノードを 1 つにして、出力が 0 と 1 のどちらに近いかで分類します。多クラス分類問題では、出力層のノードの数を分類したいクラスの数と一致させ、出力の数値が大きいほど、そのクラスだと予測している度合いが大きいとして分類します。

それでは、出力層で使われる活性化関数の話に戻りましょう。回帰問題では、恒等関数が使われることが多いです。ニューラルネットワークで関数の近似ができるので、出力の分布をわざわざ変えたくないためです。2 値分類問題では、シグモイド関数が用いられることが多いです。シグモイド関数を使う理由は、出力値が [0,1] になり、出力値を確率として解釈できることです。ちなみに、シグモイド関数はロジティック回帰などにも用いられます。

多クラス分類問題では主にソフトマックス関数が使われます。n 種類に分類するとして、k 個目のクラスのノードに対して、活性化関数に渡す前の値を a_k、活性化関数を渡した後の値を y_k とすると、ソフトマックス関数は、

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)} \tag{1.5}$$

と表されます。これを見て奇妙に思った方もいらっしゃるかもしれません。この活性化関数は、全ての出力層の値を用います。ソフトマックス関数の大きな特徴は、$\sum_{i=1}^{n} y_i = 1$ である点です。そのため、ソフトマックス関数は、出力値を確率に変える関数だとみなされることもあります。

1.3　ニューラルネットワークの学習

この節ではニューラルネットワークの学習について説明します。ニューラルネットワークは、重みやバイアスなどのパラメータの変化によって出力が変わるのでした。ニューラルネットワークの学習とは、重みやバイアスなどのパラメータを、与えたデータに合うように調整することです。学習には、主に、教師あり学習、教師なし学習、強化学習がありますが、ここでは最もわかりやすい教師あり学習について説明します。

1.3.1　教師あり学習

教師あり学習とは、データと正解ラベルをセットとして与えて、データを入力したときにニューラルネットワークが正解ラベルを出力するようにパラメータを調整する方法です。ここでは教師あり学習の例として、手書き数字認識の問題を考えます。手書き数字認識とは、図 1.4 のような手書き数字の画像が 0〜9 のどの数字を表しているのかを判別するという問題です。このとき与えられるデータセットは、手書き数字の画像とその正解を表す数字です。

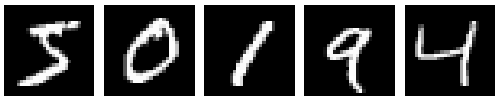

図 1.4　手書き数字のデータセット [1] より引用

データセットの画像は、28 × 28 の計 784 の画素から構成されていて、それぞれの画素が、濃淡を表す数字を持っているので、入力は 784 個の数字になります。そして出力は出力層の活性化関数をソフトマックス関数にし、出力の個数を 10 個にして、それぞれの出力の値が、実際にその画像が 0〜9 の数字でなる確率を表すようにします。例えば [0.05, 0.04, 0.80, 0.01, 0.01, 0.03, 0.02, 0.01, 0.02, 0.01] という出力は入力した画像が 0 を表している確率が 0.05、1 を表している確率が 0.04、2 を表している確率が 0.80、・・・ということを表しています。最終的にはこの確率が最も高いものを結果として出すので、目標は入力画像に対応する数字の確率を高くすることです。

1.3.2　訓練データとテストデータ

学習の方法を説明する前に、ニューラルネットワークの学習で使われるデータをどのように扱うかについて説明します。ニューラルネットワークの学習では、データは主に**訓練データ**と**テストデータ**の 2 つに分けられます。名前の通り、訓練データで学習を行い、テストデータでどれだけ学習ができているかをチェックします。データをこのように 2 つに分ける理由は、ニューラルネットワークの未知のデータに対応する能力を評価するためです。未知のデータに対して正しく推定できる能力のことを**汎化性能**といい、ニューラルネットワークの学習の最終的な目標は、この汎化性能を上げることです。もし、訓練データに対する能力だけで評価を行ってしまうと、訓練データは正解できるが、他のデータでは正解できないという状態になってしまうかもしれません。このように訓練データに過度に対応している状態を**過学習**といい、この状態になっているモデルは、未知のデータに対応できないのでよいモデルとはいえません。そこで、学習時には用いていないデータ、つまりテストデータをどれだけ正しく推定できるかという方法をとることで、ニューラルネットワークの汎化性能を正しく評価します。

1.4　損失関数

次に、ニューラルネットワークが学習する方法について説明します。学習は出力が正解に近づくためにパラメータを最適化することですが、出力と正解の近さはどのように表せばよいでしょうか。そこで、**損失関数**と呼ばれる、出力がどれだけ正解から離れているかを表す指標を用います。つまり、ニューラルネットワークの学習はこの出力と正解の離れ

ている度合いを表す損失関数を最小化することを目標とします。

1.4.1 損失関数の例

損失関数の代表例として、**二乗和誤差**があり、主に回帰問題の損失関数として使われています。これは式 (1.6) で与えられます。

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2 \tag{1.6}$$

ここで、y_k はニューラルネットワークの出力、t_k は訓練データの正解ラベルを表しています。二乗和誤差はニューラルネットワークの出力と訓練データの各要素の差の二乗の総和を表しています。正解と出力が離れているときに大きな値を取り、近いときに小さい値を取るので損失関数として適切であることがわかります。

損失関数の他の例として、**交差エントロピー**というものがあり、分類問題でよく使われます。交差エントロピーは式 (1.7) で与えられます。

$$E = -\sum_k t_k \log y_k \tag{1.7}$$

この式でも y_k はニューラルネットワークの出力、t_k は訓練データの正解ラベルを表しています。交差エントロピーは一見すると、損失関数として正しく機能するのか判断できないので、詳しく見てみましょう。

まず、交差エントロピーが主に用いられる分類問題では、正解ラベルは正解の値が 1 で、他の値が 0 である one-hot ベクトルと呼ばれる形式で与えられることが多いです。例えば、さきほど見た手書き数字認識の問題では 0 が正解の時、t =[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] のように与えられます。よって、式 (1.7) は正解となるインデックスを m とすると、

$$E = -\log y_m \tag{1.8}$$

のようになります。つまり、正解となるインデックスの出力の対数にマイナスを書けた値を計算しています。$-\log x$ は 0 から 1 の区間では、値が 0 に近いほど大きな値、1 に近いほど小さな値を取るので、正解のインデックスの出力が大きいとき、値が大きくなり、出力が小さいとき、値が小さくなります。このように正解に近いときは小さな値、離れているときは大きな値をとるため損失関数として適切だとわかります。

以上紹介した損失関数は、データ 1 つに対する損失関数です。一方、ニューラルネットワークの学習では、多くのデータで学習を行うので、データの数が N のときの損失関数は、N 個の損失関数の平均を取って、

$$E_N = \frac{1}{N}\sum_n E_n \tag{1.9}$$

となります。ただ、一般に学習に用いるデータの数は多く、大きな計算コストがかかってしまいます。そこで、全データの中から、一部のデータを選んで学習をします。この選ばれた一部のデータのことを**ミニバッチ**といい、ミニバッチを用いた学習を**ミニバッチ学習**といいます。一般にミニバッチは無作為に選びます。

1.4.2　正解率を使わない理由

損失関数を用いる理由は、ニューラルネットワークの出力と正解の近さを表すためでした。しかし、目標はニューラルネットワークの正解率をあげることなので、ニューラルネットワークの出力の正解率を指標にして、それを最大化するようにパラメータを最適化するほうが自然に思われます。

では、なぜ正解率を使わず、損失関数を導入するのでしょうか。その理由はパラメータの更新方法の指標として損失関数の方が適切であるからです。詳しくは次節で紹介しますが、ニューラルネットワークの学習では、最適なパラメータを探すときに、1 つのパラメータだけを微少に変化させたことによる損失関数の微少変化、つまり偏微分を調べます。偏微分の値が正のとき、つまり損失関数が大きくなるときは、そのパラメータを負の方向に変化させ、偏微分の値が負の時にはそのパラメータを正の方向へ変化させることで損失関数を小さくすることができます。正解率を指標として用いない理由はほとんどの場所で偏微分が 0 になってしまうからです。

例えば、手書き数字認識の場合をみてみましょう。簡単のためにデータ 1 つのときについて考えます。与えられたデータの正解ラベルが 2、ニューラルネットワークの出力の、正解が 2 である確率を表す値が 0.80 であり、ある 1 つのパラメータを少し大きくすると、その値が 0.80 から 0.84 に変わったとしましょう。パラメータが変化した結果として、正解のインデックスの値が大きくなっているので、出力としては正解に近くなっていますが、最終的な出力はそのパラメータを変える前も変えた後も 2 のままなので、正解率は変化しません。よって、ある 1 つのパラメータを変化させたときの正解率の変化が 0、つまり偏微分が 0 なので、どのようにそのパラメータを更新すればよいのかがわからないのです。

それに対して、損失関数として交差エントロピーを使ったときはどうでしょうか。パラメータを少し大きくする前は $E = -\log 0.80 \approx 0.223$ で、変化させた後は $E = -\log 0.84 \approx 0.174$ となります。損失関数の値の減少から、出力が正解に近くなっていることを判別することができ、そのパラメータを大きくすればよいことがわかります。

このように損失関数はパラメータをわずかに変化させたときの出力の変化を判別できるので、ニューラルネットワークの学習における指標として正解率ではなく損失関数が用いられています。

1.5 勾配法

前節までで述べたとおり、ニューラルネットワークを構築する際には、「学習」すなわち「最適パラメータを獲得する」ということをします。では、“最適な” パラメータとは、なんでしょうか？それは損失関数を最小にするパラメータのことです。

本節では、その最適パラメータを探す方法のうち現在の主流なアイディアのひとつを紹介します。

ニューラルネットワークを考える際に扱う損失関数の形は非常に複雑で、どのようにパラメータをとれば最小化できるのかの検討もつきません。そこで、損失関数を最小化するパラメータをあてる (ピンポイントで指定する) のではなく、パラメータを探す (更新を繰り返して最適パラメータに近づけていく) ということをします。すなわち、最初に適当にパラメータを設定して、何度も更新を加えて、損失関数を最小化する “最適な” パラメータを獲得するということです。

そこで、どのようにパラメータを更新すれば効率良く損失関数を最小化できるのでしょうか？？

その更新アイディアのひとつが、『**勾配法**』です。

勾配は、n 変数のスカラー値関数 $f(x_1, x_2, \cdots, x_n)$ に対して、

$$\frac{\partial f}{\partial \boldsymbol{x}} = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}) \tag{1.10}$$

で与えられます。また勾配は $\mathrm{grad} f$ や、∇f とも書きます。

勾配には、勾配方向への移動が関数を最も大きく増加させるという性質があります。いま、点 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ 上にいて、そこから、$d\boldsymbol{x} = (dx_1, dx_2, \cdots, dx_n)$ 移動することを考えます。すると、関数の値の変化は、

$$\begin{aligned} df &= f(\boldsymbol{x} + d\boldsymbol{x}) - f(\boldsymbol{x}) \\ &= f(x_1 + dx_1, x_2 + dx_2, \cdots, x_n + dx_n) - f(x_1, x_2, \cdots, x_n) \end{aligned} \tag{1.11}$$

となります。式 (1.11) の第 1 項目を一次の項までテイラー展開して計算すると、

$$\begin{aligned} df &= f(x_1, x_2, \cdots, x_n) + \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \cdots + \frac{\partial f}{\partial x_n} dx_n - f(x_1, x_2, \cdots, x_n) \\ &= \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \cdots + \frac{\partial f}{\partial x_n} dx_n \\ &= \frac{\partial f}{\partial \boldsymbol{x}} \cdot d\boldsymbol{x} \\ &= \| \frac{\partial f}{\partial \boldsymbol{x}} \| \| d\boldsymbol{x} \| \cos\theta \end{aligned} \tag{1.12}$$

となります。ここで、θ は勾配方向と、移動する方向のなす角です。式 (1.12) から、関数の変化 df が最大となるのは、$\theta = 0$ のとき、つまり、移動する方向が勾配方向であるときということわかります。

勾配法では、この関数の勾配方向への移動が関数を最も大きく増加させるという性質、すなわち、損失関数の勾配の “逆” 方向への移動が損失関数を最も “減らす” 移動 (パラメータの更新) であるという性質を用いて、パラメータを更新します。

更新の内容を数式で表すと以下の通りです。

$$\boldsymbol{x}_{\text{更新後}} = \boldsymbol{x}_{\text{更新前}} - \alpha \frac{\partial f}{\partial \boldsymbol{x}} \tag{1.13}$$

ここで α は『**学習率**』といい、更新方向にどのくらい更新するのかを表すパラメータです。

> 普通のパラメータ-重みやバイアス-は損失関数を最小化する過程で生成される内生的な変数であるのに対して、学習率はアルゴリズムの実装者が外生的に与えてやるものです。そういった、実装者が決める要素 (パラメータ) のことをハイパーパラメータと言います。

先ほどもお話しした通り、(座標系で見た時に) 損失関数はお茶碗のようなシンプルな形ではなく複雑です。勾配の方向に一度移動すれば最適なパラメータが見つかるわけではありません。つまり、何度も更新を繰り返す必要があります。勾配方向に学習率の分だけ移動しては、移動した先で勾配を計算し、勾配方向にパラメータを更新しては、また、勾配の計算・・・これを繰り返すことで極小値に近づけていきます。

さて、最後に、実際に具体例を用いて勾配法がどのようなイメージなのか説明します。今回はわかりやすくするために損失関数を簡易化して 2 次元の凸関数だった時を考えます。更新をするイメージは図 1.5、図 1.6 の通りです。

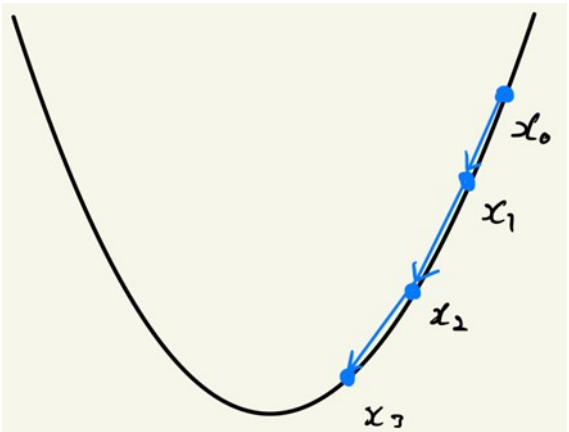

図 1.5

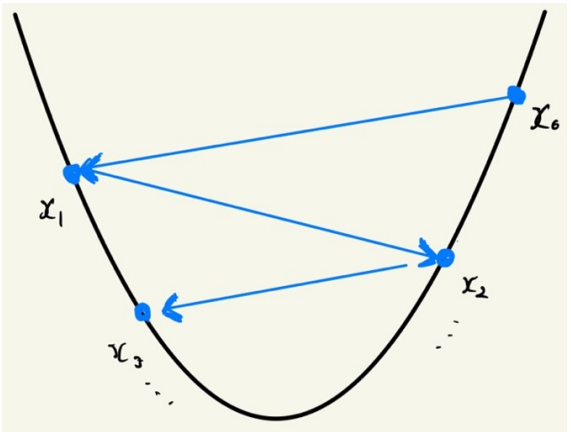

図 1.6

ある x_0 という点から始めて、$x_0 \to x_1 \to x_2$ と極小値に向かって更新をしています。もし通り過ぎてしまった場合でも、極小値に近づくように更新方向の向きが代わります。(数式からも理解できると思います。) ここで勘のいい方は気付かれたかもしれませんが、更新時の学習率が大きすぎたり小さすぎたりすると学習がうまくいきません。大きすぎると、極小値を大きく通り過ぎてしまい場合によっては更新によって発散してしまう可能性もあります。(図 1.7) 逆に、小さすぎると、更新回数が大きくなり更新させるだけで朝を迎えてしまうことになります。(図 1.8)

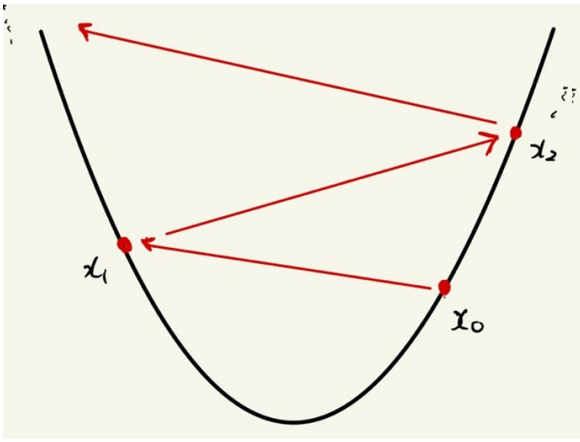

図 1.7　学習率が大きすぎる場合

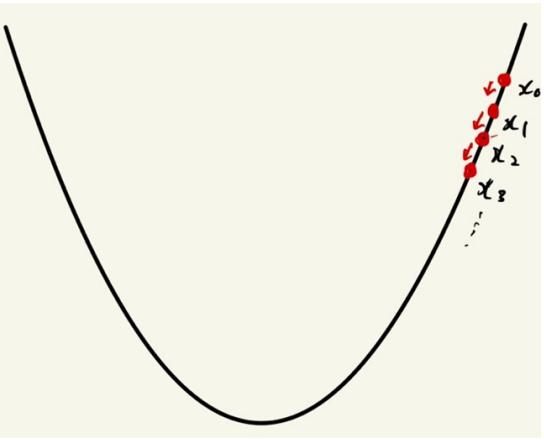

図 1.8　学習率が小さすぎる場合

また、学習率の決定方法は、いくつか試してみて最も上手くいったものを採用するという方法をとります。(一般にハイパーパラメータは同じような採用方法をとります。)

1.6　誤差逆伝播法

1.6.1　勾配の計算

勾配を使用することで損失関数の値を最小化し、「学習」ができるということがわかりました。そこで問題になってくるのが勾配の計算です。

簡単のため、2 変数関数 $f(x, y)$ を考えます。勾配を求めるためには x, y での偏微分の値を求める必要がありますが、例えば x での偏微分は式 (1.14) で表されます。

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h, y) - f(x, y)}{h} \tag{1.14}$$

しかしここで問題になるのが、普通コンピュータでは極限の計算ができないという点です。最近だと積分や極限の計算をしてくれる Web サイトやアプリがありますが、そう

いったものはコンピュータでも積分や極限の計算ができるよう特殊な工夫をしています。とにかく、コンピュータで極限の計算をするのは実はかなり難しいのです。

1.6.2　数値微分

ではどのような工夫をすれば勾配を計算できるでしょうか？例えば、愚直に h を小さい値 (例えば 0.0001) に設定して式 (1.15) を計算することで、偏微分の値を近似することができます。これは**数値微分**と呼ばれる方法で、コンピュータで微分値を計算するときによく使われる方法です。

$$\frac{\partial f}{\partial x} \sim \frac{f(x+h, y) - f(x, y)}{h} \tag{1.15}$$

しかし、Deep Learning に数値微分を用いるのは危険です。まず、Deep Learning ではパラメータ 1 つ 1 つについて偏微分する必要があります。すなわちパラメータ数と同じ数だけ式 (1.15) を計算する必要があります。ニューラルネットでは何万、何億のパラメータを持つため、いくらコンピュータと言えどもその計算には時間がかかってしまいます。

また、数値微分はあくまで近似で、誤差が生じるという問題もあります。式 (1.15) において h の値を小さくすれば誤差も小さくできますが、オーバーフローやアンダーフロー (コンピュータで扱える数値の限界を超えてしまって計算結果がおかしくなる) を起こしてしまいます。h をどんな値に設定するべきかはどんな計算を行うかによっても変わるため、ちょうどいい h の値を決めることは難しいのです。

1.6.3　誤差逆伝播法

そのような理由から、Deep Learning では**誤差逆伝播法**という方法が用いられます。誤差逆伝播法は、

- 何回も同じ計算を行うことを防ぎ、効率的に偏微分を計算する。
- (理論上) 誤差が生じない。

といった特徴を持ちます。

誤差逆伝播法の原理を説明するために、以下のシンプルな関数を考えます。

$$\begin{aligned} f_1(x) &= a_1 x + b_1 \\ f_2(x) &= a_2 f_1(x) + b_2 \\ y &= f_2(x) \end{aligned} \tag{1.16}$$

今求めたいのは、y を各パラメータ a_1, a_2, b_1, b_2 で偏微分した値です。まず、a_2, b_2 に対

する偏微分は見ただけで求まります。$\partial y/\partial a_2 = f_1(x), \partial y/\partial b_2 = 1$ です。では a_1 の方はどうでしょうか？偏微分の式を変換してみましょう。

$$\frac{\partial y}{\partial a_1} = \frac{\partial y}{\partial f_1}\frac{\partial f_1}{\partial a_1}$$

するとどうでしょうか。$y = f_2(x)$ であることを考えれば、$\partial y/\partial f_1, \partial f_1/\partial a_1$ は式を見ただけで求まります。それぞれ a_2, x です。a_1 での偏微分の値はその積ですから、$a_2 x$ ということになります。

b_1 についても同じように求められます。

$$\frac{\partial y}{\partial b_1} = \frac{\partial y}{\partial f_1}\frac{\partial f_1}{\partial b_1}$$

さて、ここで注目すべきは a_1 での偏微分の際に計算した $\partial y/\partial f_1$ の値が再利用できるという点です。ここまでは「見ただけでわかる」という表現をしてきましたが、実際にはコンピュータ上で計算するわけですからいくらか時間がかかります。その点を踏まえると、$\partial y/\partial f_1$ の値を再利用できるのは時間効率が非常に良いことになります。実際のニューラルネットワークではパラメータが何百×何百の行列で、$f_1(x), f_2(x)$ の階層構造も何十と重なることを考えると、この「値の再利用」は大きな時間短縮をもたらすことが分かっていただけると思います。

また数値微分のように近似計算も行っていないため、原理上は誤差が発生しないことになります。(もちろんコンピュータで計算する以上数値誤差は発生しますが、問題ない場合がほとんどです。)

1.6.4 誤差逆伝播法の実装例

ある縦ベクトル $\mathbf{x}$ を 1 次変換した縦ベクトル $\mathbf{x}'$ と $\mathbf{y}$ との二乗和誤差を考えてみましょう。なおベクトルの長さは全て N とします。

$$\mathbf{x}' = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{1.17}$$

$$L = \frac{1}{2}||\mathbf{x}' - \mathbf{y}||^{\mathbf{2}}_{\mathbf{2}} \tag{1.18}$$

これを各要素について書き表せば、以下のようになります。

$$x'_i = \sum_k^N w_{ik}x_k + b_i \tag{1.19}$$

$$L = \frac{1}{2}\sum_i^N (x'_i - y_i)^2 \tag{1.20}$$

では各要素についての偏微分を考えていきましょう。

まず x'_i での偏微分を考えると、

$$\frac{\partial L}{\partial x'_i} = x'_i - y_i \tag{1.21}$$

となります。ここから、w_{ik}, x_k, b_i それぞれの偏微分が求まります。

b_i での偏微分を考えると、

$$\frac{\partial L}{\partial b_i} = \sum_j^N \frac{\partial L}{\partial x'_j}\frac{\partial x'_j}{\partial b_i} \tag{1.22}$$

$j \neq i$ の時には x_j の計算に b_i が現れないことを考えると、$j \neq i$ の時 $\partial x'_j / \partial b_i = 0$ となります。

$$\begin{aligned}\frac{\partial L}{\partial b_i} &= \sum_j^N \frac{\partial L}{\partial x'_j}\frac{\partial x'_j}{\partial b_i} \\ &= \frac{\partial L}{\partial x'_i}\frac{\partial x'_i}{\partial b_i} \\ &= (x'_i - y_i)\cdot 1 \\ &= x'_i - y_i\end{aligned} \tag{1.23}$$

よってベクトル形式で書けば以下のようになります。

$$\frac{\partial L}{\partial \mathbf{b}} = \mathbf{x}' - \mathbf{y} \tag{1.24}$$

次に w_{ik} で偏微分すると、

$$\frac{\partial L}{\partial w_{ik}} = \sum_j^N \frac{\partial L}{\partial x'_j}\frac{\partial x'_j}{\partial w_{ik}} \tag{1.25}$$

となり、b_i で偏微分した時と同様、$i \neq j$ の時に $\partial x'_j / \partial w_{ik} = 0$ となるので、

$$\begin{aligned}\frac{\partial L}{\partial w_{ik}} &= \sum_j^N \frac{\partial L}{\partial x'_j}\frac{\partial x'_j}{\partial w_{ik}} \\ &= \frac{\partial L}{\partial x'_i}\frac{\partial x'_i}{\partial w_{ik}} \\ &= (x'_i - y_i)\cdot x_k \\ &= (x'_i - y_i)x_k\end{aligned} \tag{1.26}$$

よってベクトルを使って書けば以下のようになります。

$$\frac{\partial L}{\partial \mathbf{W}} = (\mathbf{x}' - \mathbf{y})\mathbf{x}^{\mathbf{T}} \tag{1.27}$$

ここで、$\frac{\partial L}{\partial \mathbf{W}}$ は成分表示すると、

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \cdots & \frac{\partial L}{\partial w_{1n}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \cdots & \frac{\partial L}{\partial w_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{n1}} & \frac{\partial L}{\partial w_{n2}} & \cdots & \frac{\partial L}{\partial w_{nn}} \end{pmatrix} \tag{1.28}$$

となります。

最後に x_k で偏微分すると、

$$\frac{\partial L}{\partial x_k} = \sum_j^N \frac{\partial L}{\partial x'_j} \frac{\partial x'_j}{\partial x_k} \tag{1.29}$$

$x'_j = \sum_k w_{jk} x_k + b_j$ だったので、$\partial x'_j / \partial x_k = w_{jk}$ となります。故に、

$$\begin{aligned} \frac{\partial L}{\partial x_k} &= \sum_j^N \frac{\partial L}{\partial x'_j} \frac{\partial x'_j}{\partial x_k} \\ &= \sum_j^N (x'_j - y_j) \cdot w_{jk} \\ &= \sum_j^N (x'_j - y_j) w_{jk} \end{aligned} \tag{1.30}$$

よって行列とベクトルを使って書けば以下のようになります。

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W^T}(\mathbf{x}' - \mathbf{y}) \tag{1.31}$$

これで各値での二乗和誤差の偏微分の値が求まりました。それぞれの結果ををもう一度書くと、

$$\frac{\partial L}{\partial \mathbf{b}} = \mathbf{x}' - \mathbf{y}, \quad \frac{\partial \mathbf{L}}{\partial \mathbf{W}} = (\mathbf{x}' - \mathbf{y})\mathbf{x^T}, \quad \frac{\partial \mathbf{L}}{\partial \mathbf{x}} = \mathbf{W^T}(\mathbf{x}' - \mathbf{y})$$

です。手順をまとめると、まず L を $\mathbf{x}'$ で偏微分した値、$\mathbf{x}' - \mathbf{y}$ を計算します。次に $\mathbf{x}' = \mathbf{W}\mathbf{x} + \mathbf{b}$ の、$\mathbf{b}$ のように、$\mathbf{x}$ から $\mathbf{x}'$ に変換するときに和がとられるものは、$\mathbf{x}' - \mathbf{y}$ に 1 をかけ、$\mathbf{W}$ や $\mathbf{x}$ のように積をとるものは、$\mathbf{x}' - \mathbf{y}$ に積を取る相手の転置をかけるように実装すればよいということがわかります。

今は、変換の回数が一回、つまり層の数がひとつの場合でしたが、深層学習のように層が深くなったとしても同じように実装ができます。まず、損失関数を最終層の入力変数で偏微分した値を求め、次に、その値を使って、最終層の 1 つ前の層の入力変数で偏微分した値を求めます。これを繰り返していくと、最終的に第一層でのパラメータで損失関数を

偏微分した値も求まります。このように、出力層から入力層へと、データを入力するときは逆向きに偏微分の値を伝搬させていくので、誤差逆伝搬法とよばれています。

PyTorch 等のライブラリでは、誤差逆伝播法と四則演算をはじめとする各関数の微分は既に定義されていることがほとんどです。ライブラリを使えば、これらの計算も自動で行ってくれて安心です。

第 2 章

BERT の解説と実用上の Tips

2.1 BERT とは

BERT とはざっくりいうと

- 自然言語処理の深層学習モデル
- 2018 年に発表、11 個の自然言語処理タスクで当時の SoTA（最高値）を達成
- 様々な派生が提案されており、画像や音声のタスクにも派生モデルあり

技術的には、

- Transformer の Encoder をベースにしたモデル
- Google や研究所などが事前学習したモデルを、使用者が fine-tuning することで様々なタスクに対応可能

Transformer は簡単に言うと

- 翻訳などの自然言語処理、時系列データ処理に使われるモデル
- 並列計算が可能で訓練時間が短い
- Encoder と Decoder からなる

イメージとしては、Encoder で既知の周辺情報を取り入れて、Decoder で未知の別の情報に変換する感じ。

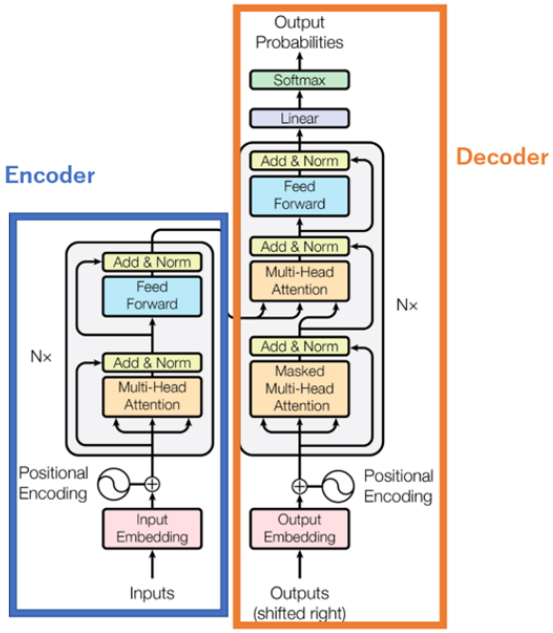

Figure 1: The Transformer - model architecture.

図 2.1

この Encoder を 12 または 24 層重ねたものが BERT の構造になる。

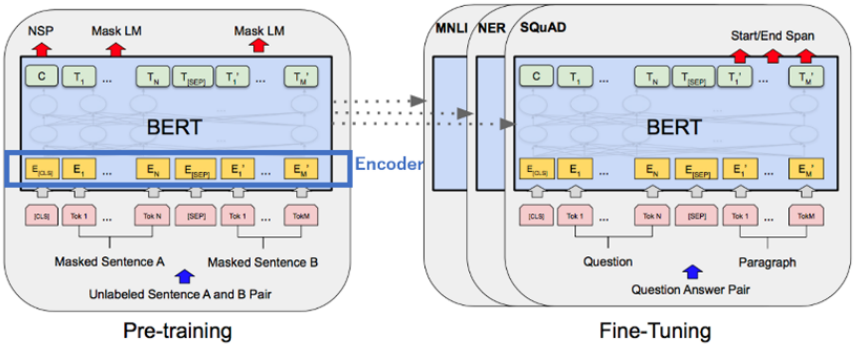

Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

図 2.2

事前学習として、文章中の単語穴埋め問題と 2 文間のつながり有無問題を学習させる。事前学習は学習時間やデータ数が非常に多くいるので、Google や研究所が公開しているものを利用するのが一般的。

事前学習の後、BERT に全結合層を加えて文書分類、固有表現抽出などの個別タスクで学習（fine-tuning）して利用する。

2.2 BERT で解かれる主なタスク

- 文書分類・・・コメントのポジティブ／ネガティブ判断、文章間の類似度判断、文章の作者分類
- 固有表現抽出・・・人名の抽出、批判的な言葉の抽出

など、fine-tuning により文書分類や固有表現抽出では自作の幅広い問題設定に対応可能。直接的に文章生成系、翻訳タスクには用いられない。

→BERT は文章を encode するだけなので、1 層追加 +fine-tuning のみで新たな文章を生成するのには向かない。（encoder 部分として使用するのは可能）

2.3 実装の手順

TensorFlow Hub [2] や Hugging Face [3] にあるトークナイザー（文章を分割し BERT 入力のトークンに変換するもの）と事前学習モデルを用いることで、Keras や Pytorch のモデルとして fine-tuning や推論ができる。

2.4 実装上の苦労

- 実行時間、メモリ
  1 文章を入力したとき、タスクにもよるが CPU（Intel Core-i9（10 コア 20 スレッド））で 0.5 秒くらいかかり、GPU（GeForce RTX 2080 Ti）で 0.05 秒くらいかかる 。GPU（GeForce RTX 2080 Ti）使用時は GPU メモリを 6GB 程度使用する。
  複数タスクのために複数モデルを同時に使用したり、入力文章が多くなったりすると、コンピュータへの負荷が大きくなるため注意。

- 入力文章の長さ制限
  Transformer 内の Attention 機構により、BERT への入力は固定長となり、最大入力トークン数は 512 となる。
  また、最大入力トークン数を長くするほどメモリ使用量も大きくなり、例えば Google Colab では 128 を超えた最大入力トークン数を指定しようとするとメモリ不足となり実行できない。
  そのため、小説など長い文章を入力するには、冒頭部分のみ使用するなどの前処理を施す必要がある。

- 固有表現抽出の難しさ
  固有表現抽出では、各トークンがどの固有表現に相当するか分類し、その連続部分を抜きだすことで表現を抽出する。
  そのため、抜き出したい表現の一部が別クラスに分類されることで、抽出表現が途切れ途切れになってしまったり、不完全な表現になってしまったりすることがある。

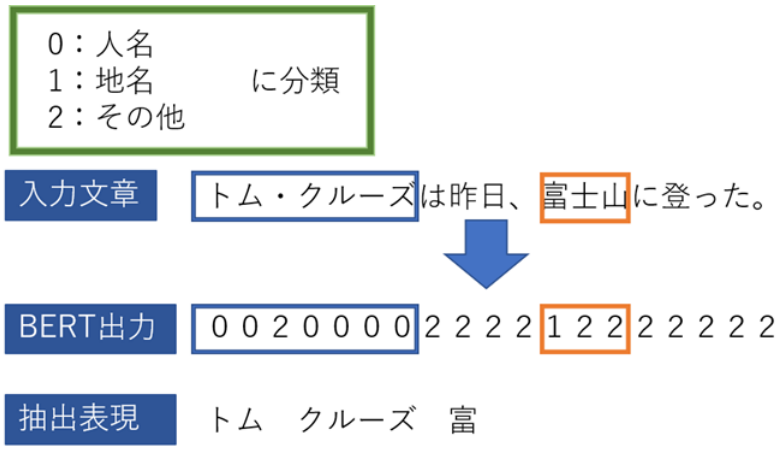

図 2.3

特に、日本語用の BERT のトークナイザーでは文章がほぼ 1 文字単位のトークンに分割されるので、長い表現は不完全になってしまうことも多い。
うまくロジックを組んで途中の誤分類を無視することもできるが、助詞（「は」「に」など）で区切られた別単語を結合してしまうなどの可能性もあるため、完全に対処するのは難しい。

第 3 章

強化学習の方策勾配法を用いた交通制御シミュレーションの改良

3.1　強化学習、方策勾配法とは

強化学習とは

- 機械学習の分野の一種
- 環境（道路、ゲーム空間）から状態、報酬をエージェント（車、コントローラー）へ渡し、今後得られる報酬を最大化するようにエージェントの行動を学習させること

図 3.1

方策勾配法とは

- 強化学習の手法の一種
- 状態から行動を決定する関数（方策関数）を直接学習する手法⇔ Q 学習
- 各行動による将来の報酬を予測する関数を学習し、予測結果から適当なアルゴリズムで行動を選択する手法

3.2　研究背景

強化学習における報酬設計は難しく、報酬から得られる情報が不十分だと非自明な最適解ではなく自明な局所最適解に陥る可能性がある。

例えば、信号機の簡単な点灯制御の場合、

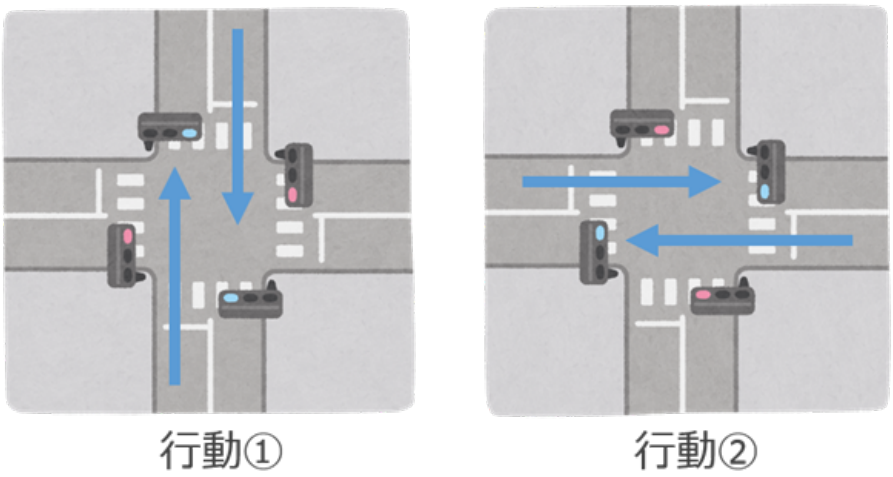

図 3.2

学習初期: ランダムに①と②を切り替え

→ 全車両の総信号待ち時間/シミュレーション=100 秒

という状況を仮定する。

そして、学習後に到達しうる状況として以下のパターン A、B を仮定する。

- パターン A: 交通量の多い方向が青になる①をほとんど選択し、まれに②を選択
  → 全車両の総信号待ち時間/1 シミュレーション=30 秒
- パターン B: 複雑な制御
  → 全車両の総信号待ち時間/1 シミュレーション=10 秒

このとき、容易に辿り着く自明な局所最適解であるパターン A に収束しがちである。ニューラルネットワークにノイズを加えて探索する行動をバラつかせることで、パターン B（非自明な最適解）に収束させたい。

こうした背景のもと、方策関数のニューラルネットワークにノイズを加えて強化学習を行った。

3.3 実験詳細

交通シミュレータ SUMO[4] を用いて、交通マップの各交差点における信号点灯パターンを制御して渋滞時間を減らすように強化学習を行った。

比較対象は RESCO[5] における実験とし、ニューラルネットワークとノイズの加え方のみ変化させて実験を行った。交通マップは RESCO[5] で実験されている 8 パターンすべてを用いた。

ニューラルネットワークは 3 通りで試し、1 つ目が純粋な方策勾配法（REINFORCE）を用いたもの、2 つ目が PPO というモデルを用いたもの（全結合層のみ）、3 つ目が RESCO[5] と同じニューラルネットワーク（PPO、2 つ目と違い畳み込み層を使用）とした。

ノイズの加え方も 3 通り用意し、1 つ目が中間層出力に正規分布のノイズをのせる、2 つ目が最終出力層の softmax 関数に温度パラメータを設定する、3 つ目が VQ-VAE における VQ の部分を用いて中間出力を離散化させる、とした。各手法においてノイズの大きさを 3 通りに変化させて実験した。

3.4 実験結果と考察

ニューラルネットワークは 3 通りの内、3 番目の RESCO[5] と同様のものが一番性能がよかった。

また、ノイズを加えた場合でも、到達した渋滞時間の少なさはあまり変わらなかった。

しかし、学習後の行動系列を PCA で可視化、及び行動パターンを実際に見ると、ノイズを加えたときに少し違いが生じる場合があった。

図 3.3 は、1 学習間隔の間にとった行動パターンの系列をベクトルと見なし、PCA で 2 次元に可視化したものである。左側がノイズなしで学習させた過程、右側がノイズありで学習させた過程になっている。モノクロ表示及び点の数の多さにより見づらくなっているが、〇で囲ったところに収束している。また、上の行が交差点が 8 つある交通マップでの結果で、下の行が交差点が 16 個ある交通マップでの結果である。other actions は同じ交通マップでのすべての実験を比較のために載せたものである。

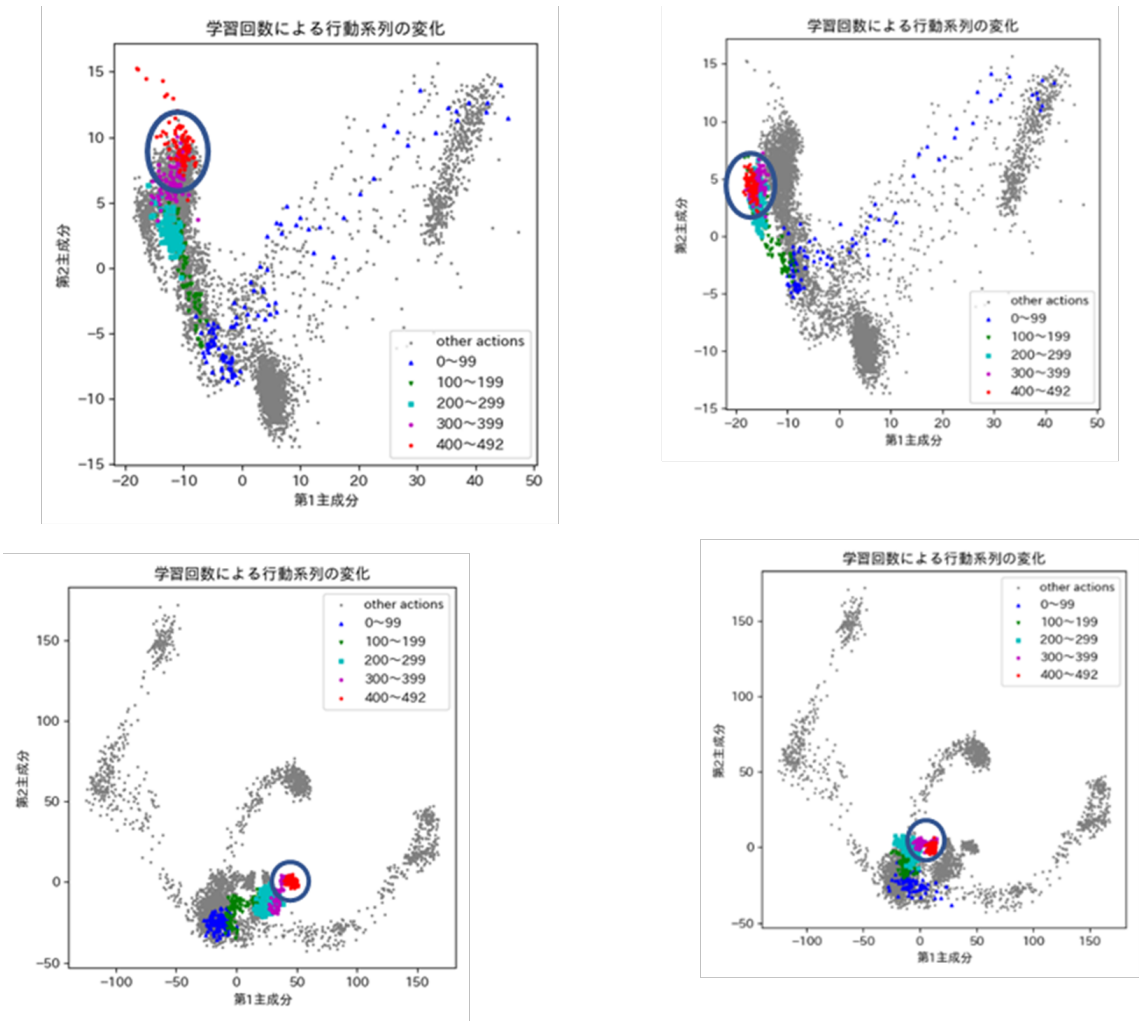

図 3.3

ノイズを加えることによって収束している行動パターンが少し変化していることがわかる。

実際に学習後の行動パターンを上の画像に対応して見ると、図 3.4 のようになる。横軸が 1 シミュレーション内での時間ステップ、縦軸がとった行動パターンである。各交差点ごとに行動をとるので、交差点 8 つのマップでは 8 つのグラフ、交差点 16 個のマップでは 16 個のグラフが表示されている。横に見て、〇で囲んだ部分が特に異なっていることがわかる。

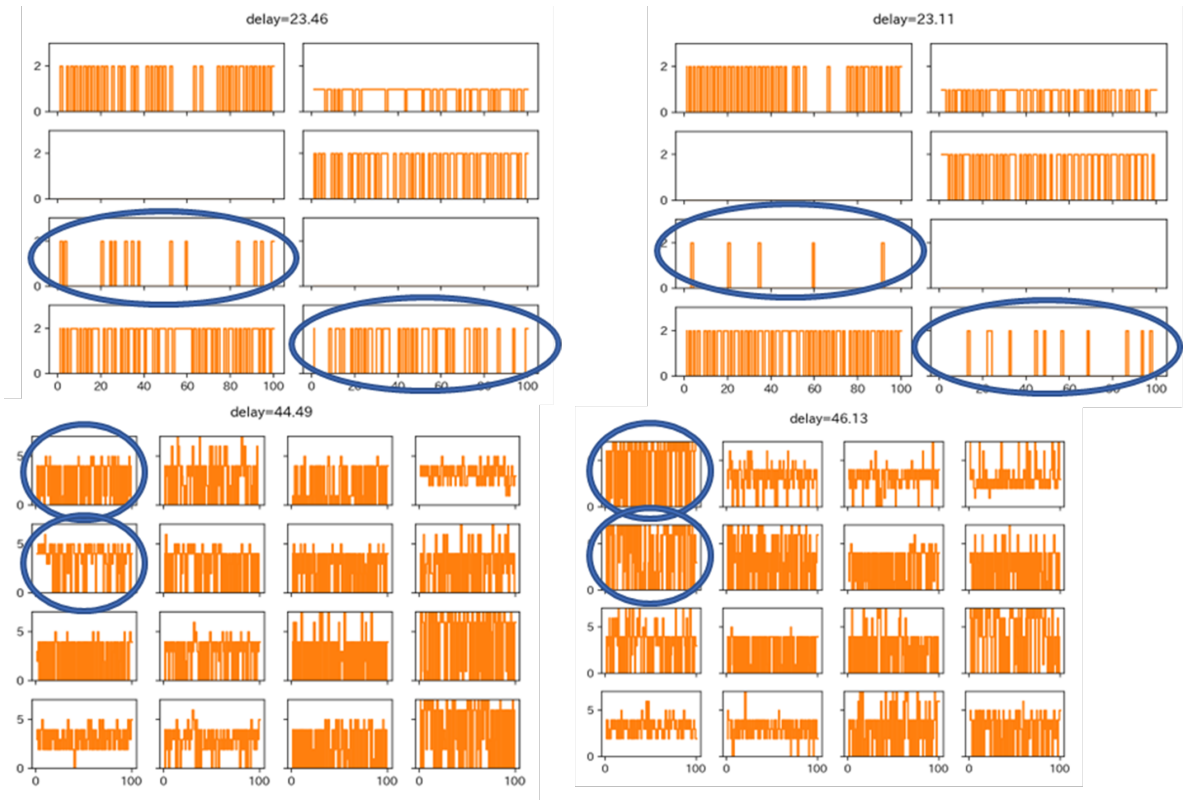

図 3.4

3.5　結論

強化学習の方策勾配法において、方策関数のニューラルネットワークにノイズを加えることでモデルを改良することを実験した。ノイズを加えることで指標値で最高値を出すことはできなかったが、ノイズの効果により同じ指標値でも違う制御状態にたどり着く可能性があることがわかった。まだまだ研究の途中であるので、さらなる試行や考察をしていきたい。

3.6　関連研究

- Noisy Network for Exploration
  `https://arxiv.org/abs/1706.10295`
  ニューラルネットワークのパラメータ自体にノイズを加え、広範囲の解の探索を行う
- Variational Empowerment as Representation Learning for Goal-Based Reinforcement Learning

https://arxiv.org/abs/2106.01404
行動とそれによる将来状態の間の相互情報量を最大化することで、広範囲の解の探索を行う

第 4 章

アニメスタイルへのリアルタイム映像変換

4.1 概要

本セクションはデモとして展示している「全身映像のリアルタイム変換」についての記事です。執筆当時はまだデモが完成していないため、途中までとなりますがご容赦ください。

本手法では、Encoder を工夫することにより画像変換手法を動画変換手法に拡張します。

4.2 手法

4.2.1 画像生成

本手法では StyleGAN2[6][7][8] をベースに用いています。図 4.1 にモデルの概要図を示します (これは StyleGAN2 ではなく StyleGAN のモデル図なのですが、ほぼ同じと考えてもらって構いません)。

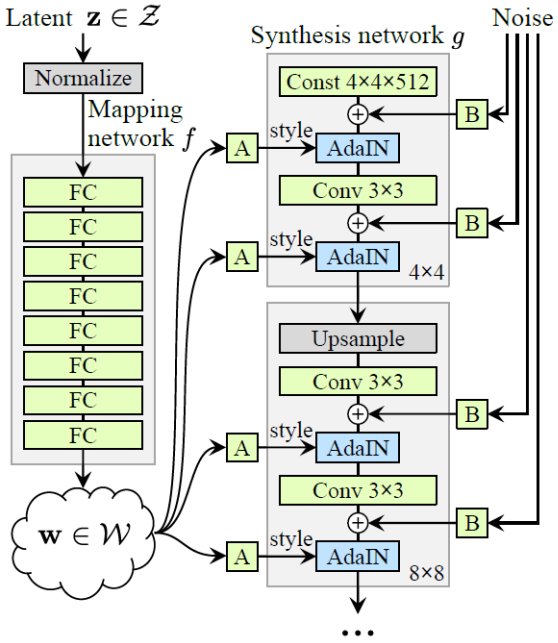

図 4.1　StyleGAN のモデル概要。[6] より引用。

StyleGAN2 では潜在変数ベクトル (図中の $\mathbf{w}$) を各層に入力します (図中の A)。[*1]生成画像の解像度が 1024 の場合は層が 18 個あるので、18 個の潜在変数を入力することになります。生成時には基本的に 18 個同じ潜在変数を用いるのですが、もちろん 17 個を同じ潜在変数として残りの 1 個を別の潜在変数にすることもできます。StyleGAN2 の特徴としては、この 1 個の異なる潜在変数を 18 層のうちどこに入力するかによって生成画像の変化の仕方が異なるという点です。(所謂 Style Mixing)

実際に生成画像を見てましょう。図 4.2 は 2 種類の潜在変数を混ぜて入力した時の生成画像です。1 段目 (Source B) と 1 列目 (Souce A) の画像だけは 1 種類の潜在変数から生成した画像です。中央のグリッドでは、例えば”Coarse styles from source B”と書いてある行は「Coarse 層には B の潜在変数を、他の層には A の潜在変数を入力して生成した画像」、という見方をします。

Coarse 層 (日本語では「粗い」; 最初の方の層を指します) では B の潜在変数を入力することで A の性別や全体的な顔の形などが B のものに変化しています。対して Fine 層 (日本語では「細かい」; 後半の層を指します) では B の潜在変数を入力しても、A の画像から背景や髪の色 (モノクロだとわかりづらいかもしれませんが…) が変わった程度です。

[*1] $\mathbf{w}$ は、正規分布からサンプリングした $\mathbf{z}$ を Mapping Network(FC は全結合層) に通して計算されます。その後最初は 4x4 の Const からスタートし Conv 等をかけ、Upsample で解像度を 2 倍の 8x8 にして同様に Conv 等…と解像度が 1024 になるまで繰り返していきます。

図 4.2　Style Mixing の生成例。[6] より引用。

このように StyleGAN2 では最初の方の層は主に形状に影響し、後半の層は主に色に影響することがわかります。

4.2.2　画像変換

StyleGAN2 をベースとした画像変換として、Toonify[9] があります。Toonify は極めてシンプルな手法です。例えば実写顔からアニメ顔への変換をするには、まず実写顔画像で StyleGAN2 を訓練した後、アニメ顔画像で fine-tuning を行うだけです。推論時には、StyleGAN2 の前半の重みを fine-tuning 前、後半の重みを fine-tuning 後のものにすることで、「実写風のアニメ顔」の画像が生成されるようになります。

図 4.3 は実際に生成した「実写風のアニメ顔」の画像です。左から順に fine-tuning 前に生成した実写顔画像、fine-tuning 後に生成したアニメ顔画像、それらを mix して生成した実写風アニメ顔画像です。全体的な構図は実写顔画像と一致していますが、色味やテクスチャはアニメ顔画像の方に一致していることがわかります。

図 4.3　Toonify の生成例。実写顔画像生成は筆者が学習させたモデルを使用。アニメ顔画像モデルは Gwern 氏のモデル [10] を使用。

4.2.3　Encoder

また画像を入力とするためには Encoder が必要になります。StyleGAN2 は 18 個の潜在変数を入力に取りますが、画像変換のためには入力画像を一旦潜在変数に変換する必要があります。そのために使われるのが Encoder で、StyleGAN2 の Encoder としては pSp[11] や e4e[12] が知られています。

pSp や e4e は入力画像から潜在変数 18 個を予測するモデルですが、例えば MegaFS[13] では入力画像の Coarse 情報をより高めるために 4x4 の特徴マップ (従来の StyleGAN2 では固定パラメータとしていた) も Encoder で推定しています。また、Pose with Style[14] では全身画像のポーズ情報保持のために 16x16 の特徴マップを推定しています。

16x16 の特徴マップ以下の計算量を低減するためにも、これに倣い 16x16 の特徴マップを Encoder で推定することにしました。16x16 の特徴マップを実写画像で学習した Encoder で生成し、それ以降はアニメ画像で fine-tuning したモデルを用いることで、実質的に Toonify を行っていることになります。

4.3　学習結果

まずは全身実写画像での学習結果です。なお解像度が 512 だとモード崩壊という現象が起こり上手く学習できなかったため、256 の解像度を採用しています。図 4.4 が生成画像です。1 段目はランダムに生成した画像、2 段目は同じシードで Truncation を 0.6 に設定して生成した画像です。Truncation とは、その値を α、潜在変数を w としたときに、潜在変数を $\alpha w + (1 - \alpha) w_{mean}$ と更新する手法のことです (w_{mean} は潜在変数の平均値)。つまり、Truncation の値を小さくすると潜在変数は平均によることになります。潜在変数を平均に寄せると生成画像が綺麗になってくれて嬉しいのですが、その代わりに生成画像が似たり寄ったりになるという欠点もあります。

図 4.4　全身実写画像の生成例。

次に全身アニメ画像での fine-tuning 結果です。同様に図 4.5 に生成画像を示します。

図 4.5　全身アニメ画像の生成例。

最後に Toonify での生成結果です。1 段目が全身実写モデルでの生成結果、2 段目が全身アニメモデルでの生成結果、3 段目がそれらを mix した Toonify での生成結果です。3 段目の変換画像は、確かに 1 段目のポーズ情報を持ち、2 段目のアニメ画像のテクスチャを持っていることがわかります。(fine-tuning しているので、そもそも 1 段目と 2 段目のポーズはほぼ同じになってしまいます)

図 4.6　全身モデルでの Toonify の生成例。

4.4　前処理

図 4.7　前処理のパイプライン。画像はインターネット上のフリー画像を使用。

さて、実際のリアルタイム変換デモでは、生成画像を安定させるために以下の前処理が必要になります。また図 4.7 に前処理のパイプラインを示します。

1. カメラに映った全身姿を正面に持ってくる
2. 背景を除去する

1 には YOLOv5[15] と MTCNN[16] を使用しています。YOLO シリーズは高速に動作する物体検出モデルで、YOLOv5 はその中でも新しいモデルです。YOLOv5 でカメラ画像から human クラスの物体を検出します。また、今実現したいのはカメラ画像の上端を人の頭に揃えることなので頭の位置も検出する必要があり、MTCNN という顔検出ライブラリを用いています。

ただし YOLOv5 も MTCNN もあくまで画像から人物領域あるいは顔領域を求めるモデルです。動画に対してもフレームごとにモデルを通せば適用できるのですが、フレームごとに領域が少しずつずれてしまいます。そして実際、動画をクロップした際にこのズレはかなり目立ってしまいます。これを防ぐために指数移動平均 (EMA; Exponential Moving Average) でクロップ領域を決めています。例えば x の指数移動平均 $\bar{x}$ はパラメータ α(1 以下の正数) を用いて次のように更新されます。

$$\bar{x} = \alpha x + (1 - \alpha)\bar{x}$$

この式を見ると、x が急激に大きくなっても $\bar{x}$ の値はそれほど大きくならない、ということがわかります。今回の場合、YOLOv5 で求めた座標が x で、実際にはその指数移動平均 $\bar{x}$ の座標でクロップします。[*2]こうすることでクロップ座標のズレは検出した座標のズレよりも小さくなり、動画にしても目立たなくなります。なお α は 0.8 に設定しています。

2 には RobustVideoMatting[17] を用いています。これは HD 解像度の画像を 1 秒間に 104 枚処理できる[*3]という驚くべき高速な手法です。リアルタイム変換のためには前処理にあまり時間をかけられないため、YOLO 同様高速なモデルを採用しました。

これらを組み合わせ前処理として実装し、Nvidia T4 上で動かしてみると約 16FPS でした。これはつまりカメラで映像を映してから前処理が終わるまで約 0.06 秒のタイムラグが発生することになります。今回はこの程度の遅延は問題ないと判断し、このパイプラインを採用しました。

4.5 実装

今回は GCP(Google Cloud Platform) 上にデモサイトを用意しました。UI の作成には Streamlit を使っています。Stremalit は映像を高速にやり取りするための WebRTC

[*2] 矩形にクロップするので、実際には左上の xy 座標と右下の xy 座標、合計 4 つの変数の EMA を取ります。

[*3] 1 枚の Nvidia GTX 1080Ti GPU 上でのスコアです。

という機能を Python から呼び出すことができます。また前処理や変換処理は Docker コンテナ内に FastAPI で API を立ち上げ、Docker Compose で各コンテナを管理しています。

4.6　まとめ

StyleGAN2 を使った実写からアニメスタイルへ変換するモデルを作成しました。また前処理を高速かつ安定して変換されるよう実装しました。StyleGAN2 の変換部分がボトルネックとなるため、今後はその部分の軽量化に注力していきたいです。

デモは KaiRA 展示ブースで体験することができます。ぜひお越しください！

第 5 章

3D KaiRA 君への姿勢推定技術の適用について

5.1 はじめに

VR ゴーグルの開発やアバターを用いた配信者の出現などにより，仮想 3 次元空間におけるサービスを提供するメタバースとその技術は注目を浴びている．メタバースを支える技術は，並列計算に優れた計算機や人の骨格を推定する姿勢推定技術，効率的な 3D 表現など多岐に渡り，導入コストは高いものとなっていた．しかしそれらの技術は着実に進化してきており，必要なコストを減らしている．例えば，姿勢推定においては，深層学習の発展により，図 5.1 のようなカメラ画像から人の関節位置を推定する技術などが現れ，導入コストを抑えることができるようになった．この結果，多くの人にとってアバターを用いたオンラインコミュニケーションに触れる機会が増加した．

一方，本研究会のオンライン・ハイブリッド勉強会では，スライドのみが画面に表示され，オンラインで聞いている人は対面の時のように聞くことが難しくなっていた．そこで本稿では，マスコットキャラクターの 3D モデルを製作し，（実装が間に合わなかったため）勉強会を中心としたそのモデルの活用方法の考察について説明する．本研究会のマスコットキャラクターである KaiRA 君は図 5.2 のような外見で，本研究会の OB によってデザインされた．本体は脳であり，その後部にはニューラルネットワークを模した鞭毛が搭載されている．

本稿では，まず，姿勢推定技術について，その後 3D のモデル表現について説明する．最後に，3D KaiRA の利用方法について説明する．

図 5.1　姿勢推定技術の例

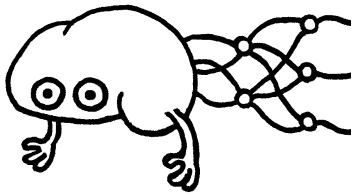

図 5.2　マスコットキャラクターの KaiRA 君

5.2　姿勢推定技術

姿勢推定技術は，センシングしたデータをもとに人の姿勢・骨格を推定する技術のことである．このセンシング手法には大きく 3 つの手法を用いられることが多い．

1）光学センサを用いた手法

図 5.3 のように，赤外線を出力する発光素子とその受光素子を一体化した機器と，光を入射された方向に反射する特別なマーカーを使用する手法である．この機器は，赤外線が射出されてから，反射し戻ってくるまでの時間を計測することでマーカーとの距離を計算するためにある．この機器とマーカーを複数用いることで様々な部位の相対的な位置を把握することができ，その位置を組み合わせることで結果的に絶対的な位置を得ることが出来るのである．この方式では高精度な関節位置を得ることが可能であるが，機器の設置コストや広い部屋が必要など環境に強く依存する．

2）慣性センサを用いた手法

図 5.4 のように，加速度センサーを関節位置に取り付けることで，その値や変化量から位置を推定する手法である．これは環境に依存しないが，精度の誤差が蓄積されやすいため時間が経つほど精度が劣化する．

3）カメラ画像を用いた手法

図 5.5 のように，機械学習を用いて，画像や動画を学習済みモデルに入力し，関節位置の座標を出力する．ベンチマークデータセットとしては”Human 3.6m”や”MPI-INF-3DHP”，”HumanEva-1”などが知れらる．機械学習モデルは，深層学習，その中でも

Transformer(ViT) を用いたものが高い精度を誇っている．これらは学習するコストがかかるが，一度学習させてしまえば，カメラと優れた計算機を用意するだけで良いので導入コストは低い．（とはいえその優れた計算機を手に入れるのは苦労するかもしれない．）

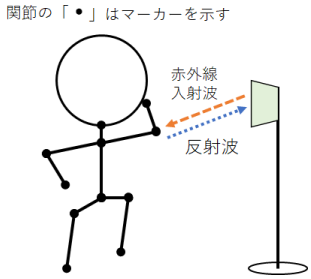

図 5.3　光学センサ方式

図 5.4　慣性センサ方式

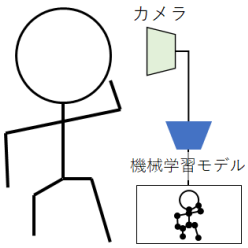

図 5.5　カメラ方式

話は少しそれるが，個人的に気になっている推定方法はスタンドアロン型 VR ゴーグルのコントローラ位置や傾きの推定である．VR ゴーグルにおける姿勢推定ではヘッドマウントディスプレイ（HMD）やコントローラは SLAM センサを用いて，その位置情報を得ている．SLAM とは Simultaneous Localization and Mapping の略称であり，自己位置推定と周囲の地図作成を同時に行うことだ．VR ゴーグルは，部屋の状況を把握しつつ，その中で自身がどこに存在するのかやコントローラはどこにあるのかということも把握することが出来るということである．限られたリソースの中で，自由度の高い推定を行うために工夫されていることが伺えて興味深い．

5.3　3D 空間における形状表現

姿勢推定技術により得られた位置情報を 3D モデルに適用する必要がある．そこで，そもそも 3D 空間における 3D モデルがどのように構成されているのか，どのようにして動かされるのかを紹介しよう．

5.3.1　3D モデルの表示

3D 空間を表現しそれを出力するデバイスは Looking Glass などの特殊なものを除き 2D として出力するディスプレイが多くを占めるだろう．3D 空間を 2D へと出力するプロセスは，大まかに図 5.6 のように説明することができる．まず 3D 空間において，形状を表す点集合データであるモデルを用意する．次にライトを用意して，映したいものをカメラで映す．このカメラで撮影したものが画像として表され，ディスプレイに表示されるという仕組みとなっている．

また，モデル形状の表現においてその方法は大きく分けて 3 つある．まずはワイヤーフ

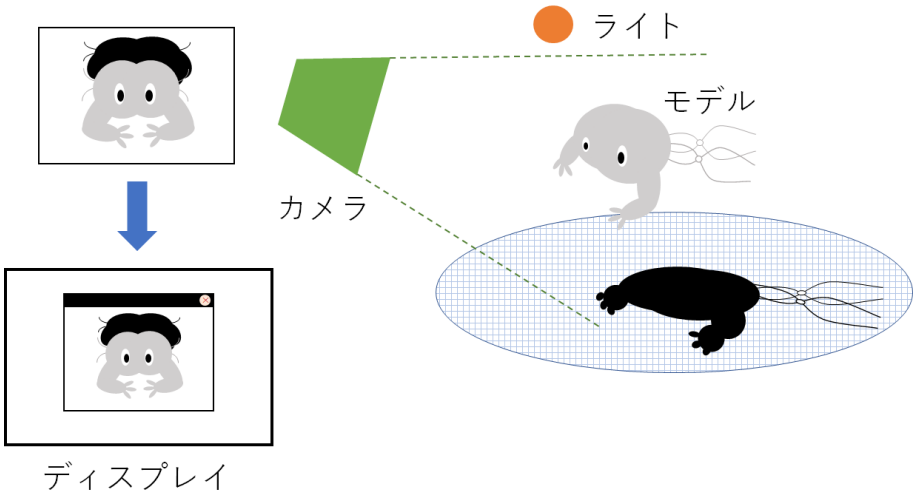

図 5.6　3D 空間をディスプレイに投影するまでの概要

レーム表現である．この表現方法は頂点同士を結ぶ線のみを記録し軽量な計算方法であるため，データアクセスやモデルの移動・変形を高速に行うことが可能となる．しかし全ての線を描画するため，奥行き関係といった立体構造を直感的に理解することは難しい．二つ目はサーフェイス表現である．これはワイヤーフレームモデルに面情報を追加した表現である．一番手前の面より奥にあるデータは表示されなくなるという陰線・面消去といった処理が行われる．最後はソリッド表現である．これはサーフェイス表現に，さらに物体の内外の情報を加えたモデルである．このソリッド表現はモデル製作時にデフォルトで用いる表現方法である．

5.3.2　キャラクターモデルの動かし方

キャラクターを動かす，つまりアニメーション製作では 3D モデル内に「ボーン」と呼ばれる骨格を組み込む．このボーンの位置に合わせてサーフェイスの形状は変化し，まるで骨があるかのように動くことが出来る．なので姿勢推定技術を用いてキャラクターを動かす場合は，推定する関節の数とボーン間の数を一致させる必要がある．

他にも，リップシンクで用いられる，サーフェイス（メッシュ）をある格子状に分割し，格子の中心に質点，辺をバネと仮定した方法や髪の動きに用いられるパーティクルを用いた手法がある．

5.4　姿勢推定技術を用いた KaiRA 君のアバター活用方法

今までの情報を用いて 3D モデルの活用方法について考えが，その前に作成したモデルを紹介する．図 5.7 に今回作成した KaiRA 君モデルを，図 5.8 にそのモデルのボーンの様子を示す．今回作成したモデルにはニューラルネットワークを模した鞭毛がない．というのも姿勢推定する際にボーンを一致させることができないからである．（物理シミュレーションで適当に動かすという考えはある）

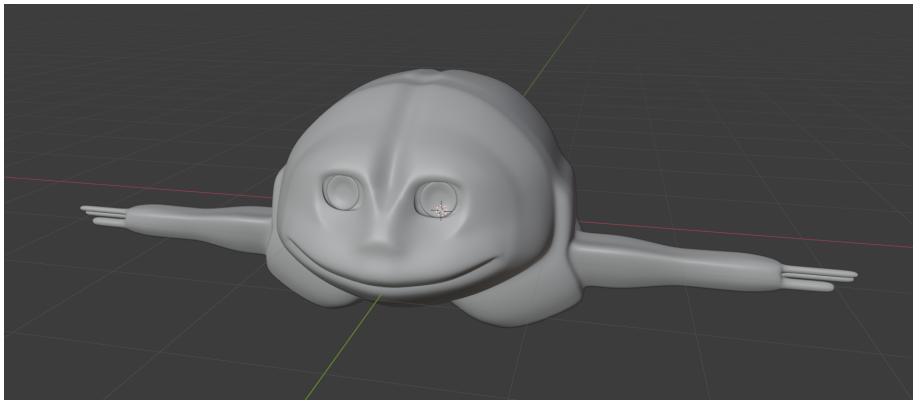

図 5.7　製作した KaiRA 君の 3D モデル

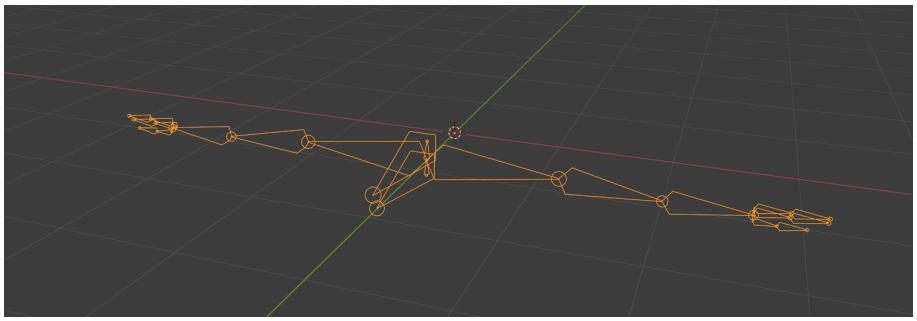

図 5.8　KaiRA 君のボーン

1）ハイブリッド勉強会における発表者アバター

3D モデルの勉強会への活用方法を考えてみる．KaiRA では毎週勉強会を行っていたり，昨年のオンライン NF では動画を上げていたりしている．その際に，参加者や視聴者により馴染みやすくするために 3D モデルを用いたい．今回は使用頻度の高そうな勉強会を想定して話を進める．

勉強会では部屋を借りて発表する．そのため光学センサを用いた姿勢推定機器などを運

搬し設置するのは難しい．したがって慣性センサを用いた手法やカメラ画像を用いた手法に限られる．さらに，毎回関節の位置にセンサを付けるのは煩わしいので，カメラ画像を用いた手法が良いだろう．そして関節の位置の推定時間は音声と映像のラグに直結するので，モデルは軽量または未来の姿勢を推定するものを選ぶ必要があると思われる．

2）KaiRA 君を用いたゲーム

次は勉強会以外に用いられそうなことについて考える．近年は Unity や Unreal Engine，Godot Engine といった無料で利用できるゲームエンジンの台頭により個人ゲーム製作が活発になってきており，特に 3D ゲームにおいては顕著であるように感じる．3D ゲームにおいて，クリエイターでない人間にとって二つ課題がある．それは，そもそも 3D モデルを作るのが難しいという点とその後アニメーションを製作するということであるだろう．前者の課題に関しては，複数の画像から 3D のフィールドを再現する NeRF といった技術により取り組まれている．後者に関しては，上記の通り姿勢推定といった手法により取り組まれている．

3D モデルを作成した今，姿勢推定によりモーションを取り，キャラクターの動きに適用してみたい．最近，本研究会内でもゲームとの相性が良い強化学習が流行っているので，何か企画があっても面白いかもしれない．

5.5　まとめ

対面やハイブリッド型の授業が増えてきた中，対面とオンラインで受ける方たちの体験する雰囲気に違いを出させないために，姿勢推定のような技術が必要となる．今回はマスコットキャラクターを 3D モデル化してその利用方法について考えた．実装が出来たら公開したいと思う．

参考文献

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. `https://yann.lecun.com/exdb/mnist/`.

[2] Google, "bert." `https://tfhub.dev/google/collections/bert/1`, 2022. Accessed on 2022-11-12.

[3] H. Face, "Models." `https://huggingface.co/models`, 2022. Accessed on 2022-11-12.

[4] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, 2018.

[5] J. Ault and G. Sharon, "Reinforcement learning benchmarks for traffic signal control," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[6] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4217–4228, 2021.

[7] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8107–8116, 2020.

[8] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training generative adversarial networks with limited data," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[9] J. N. Pinkney and D. Adler, "Resolution dependent gan interpolation for controllable image synthesis between domains," *arXiv preprint arXiv:2010.05334*, 2020.

[10] Gwern, "Making anime faces with stylegan." https://www.gwern.net/Faces, 2022. Accessed on 2022-11-11.

[11] E. Richardson, Y. Alaluf, O. Patashnik, Y. Nitzan, Y. Azar, S. Shapiro, and D. Cohen-Or, "Encoding in style: a stylegan encoder for image-to-image translation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2287–2296, 2021.

[12] O. Tov, Y. Alaluf, Y. Nitzan, O. Patashnik, and D. Cohen-Or, "Designing an encoder for stylegan image manipulation," *ACM Trans. Graph.*, vol. 40, no. 4, 2021.

[13] Y. Zhu, Q. Li, J. Wang, C. Xu, and Z. Sun, "One shot face swapping on megapixels," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4832–4842, 2021.

[14] B. Albahar, J. Lu, J. Yang, Z. Shu, E. Shechtman, and J.-B. Huang, "Pose with style: Detail-preserving pose-guided image synthesis with conditional stylegan," *ACM Trans. Graph.*, vol. 40, no. 6, 2021.

[15] G. Jocher, "ultralytics/yolov5." https://github.com/ultralytics/yolov5, 2022. Accessed on 2022-11-11.

[16] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[17] S. Lin, L. Yang, I. Saleemi, and S. Sengupta, "Robust high-resolution video matting with temporal guidance," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3132–3141, 2022.

[18] 斎藤康毅, ゼロから作る *Deep Learning Python* で学ぶディープランニングの理論と実装. オライリー・ジャパン, 2016.

[19] オミータ, "深層学習界の大前提 transformer の論文解説！ (qiita)." https://qiita.com/omiita/items/07e69aef6c156d23c538, 2020. Accessed on 2022-11-11.

[20] オミータ, "自然言語処理の王様「bert」の論文を徹底解説." https://qiita.com/omiita/items/72998858efc19a368e50, 2020. Accessed on 2022-11-11.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

サークル情報

京都大学人工知能研究会 KaiRA

代表　三宅大貴 (工学部三回生)
活動日時　毎週木曜日 18:30〜
活動場所　京都大学附属図書館三階共同研究室 5(オンラインでも参加できます)
入会資格　大学、学部、回生問わず誰でも
会費　無料
Mail　kyoto.kaira@gmail.com
Website　`https://kyoto-kaira.github.io/contact.html`
Twitter　@kyoto_kaira

活動内容

- **輪読会**
  深層学習や機械学習に関する本の輪読を行っています。本を章ごとに分けて、発表者を決め、議論して理解を深めています。
- **実装開発**
  学んだ知識をもとに、実装開発をしています。結果は成果報告会や NF で発表をしています。
- **コンペティションへの参加**
  Kaggle の機械学習のコンペティションに参加しています。KaiRA のメンバーでチームを組み、取り組むこともあります。

執筆者紹介

加藤雅大

京都大学工学部物理工学科 3 回生宇宙基礎工学コース
機械学習やデータ分析、制御に興味があります。実務としては 1 年半ほど企業のインターンで自然言語処理を行っています。研究としては現在、強化学習系のことを行っています。Qiita や Github には過去に実装した Deep Learning モデルやシミュレーションなどを載せています。
Qiita: https://qiita.com/KMASAHIRO
Github: https://github.com/KMASAHIRO

三宅大貴

工学部電気電子工学科の 3 回生です。
画像生成に興味があり、GAN 周りの実装経験があります。最近は Stable Diffusion 関係の勉強・実装もしています。
GitHub: https://github.com/KYM384
はてなブログ: https://kym384.hatenablog.com/

羽原丈博

集積回路系の修士一年生です。
コンピュータビジョンやニューラルネットワークの低消費エネルギ化の研究をしています。CG にも興味があり、最近は地形生成・シミュレーションに取り組んでいます。
Github: https://github.com/hanebarla
Qiita: https://qiita.com/barla_flap

文責

第 1 章　山下素数,　大前俊輔,　米山瑛人,　三宅大貴
第 2 章　加藤雅大
第 3 章　加藤雅大
第 4 章　三宅大貴
第 5 章　羽原丈博