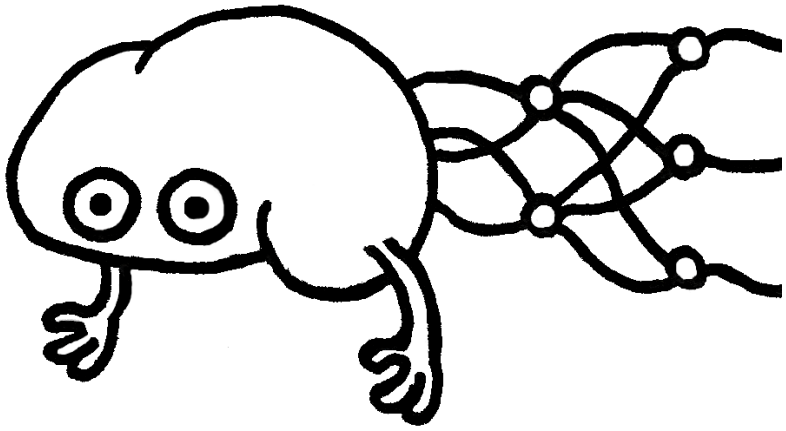
KaiRA 会誌 vol.5

京都大学人工知能研究会



目次

1　Deep Learning 入門

1.1　Deep Learning 概要

1.1.1　Deep Learning とは

Deep Learning とは機械学習手法の一種です．近年の AI ブームの中心には Deep Learning があると言えます．少し前のニュースで言えば囲碁で AI が人間に勝利したというものがありました．また身近な例でいえば，LINE で話すことができる「りんな」や「エアフレンド」，長文を要約する「ELYZA DIGEST」といったサービスや，「自撮り写真を老けさせるアプリ」「顔写真の性別を変えるアプリ」などのアプリケーションが存在します．これらもその根幹となる部分では Deep Learning が用いられています．

Deep Learning の特徴は，計算するパラメータが多いことです．もっと簡単に言えば，とてつもない数の計算を行うということです．これは画像に何が写っているかを判断したり，言語を翻訳したりなど複雑な課題を解く際に有利になります．その反面，処理に時間がかかるという欠点があります．近年の AI ブームも，コンピュータが発展してようやく Deep Learning を実装できるようになったという背景があります．そして今でも「どれだけ性能の良い AI を開発できるか？」という問題と「どれだけ性能の良いコンピュータが使えるか？」という問題は密接な関係にあります．

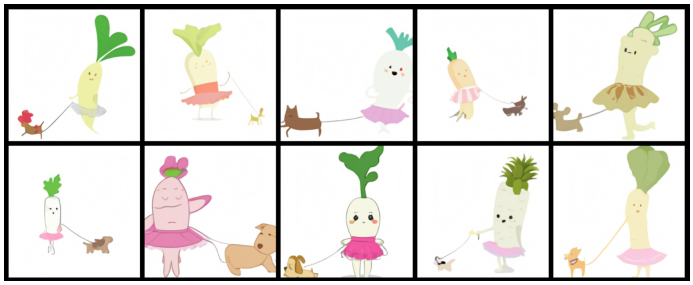
ここでは Deep Learning 入門と称して，ニューラルネットワークとその学習方法について紹介します．

1.1.2　CLIP と DALL-E

今年の Deep Learning 界隈で大きな衝撃になった AI モデルを紹介します．それは今年の 1 月に OpenAI から発表された，**CLIP**[1] と **DALL-E**[2] というモデルです．

CLIP は我々人間が使う自然言語と画像を結びつけるモデルです．具体的には，ある画像とある文章がどれだけ関係あるかを計算します．例えば冬の写真と「雪が降っている」という文章は関係性が高いと言えます．逆に，食事の写真と「赤くて大きい車」という文章は全く関係がないと言えます．CLIP が開発されたことで人間には当たり前だった言語と視覚を結びつけるという能力が機械にも可能になりました．

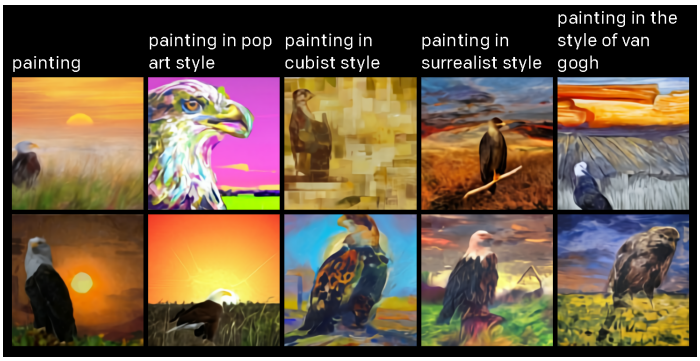
CLIP を用いて開発されたのが DALL-E です．DALL-E は文章から画像を作るモデルです．驚くべきはその多様さです．OpenAI のサイト [3] に載っている例をいくつか紹介します．なお以下の画像は全て [3] からの引用です．



an illustration of a baby daikon radish in a tutu walking a dog (チュチュを着て犬の散歩をする大根の赤ちゃんのイラスト)

「チュチュ」「犬の散歩」「赤ちゃん」「大根」という全く別の言葉を上手くまとめてイラストにしています．またこのような文章とイラストは学習データに存在しにくいと考えられますが (シュールすぎるので)，それでもこの文章を見た時に人間が想像するのと同じようなイラストを生成できています．

an extreme close-up view of a capybara sitting in a field (近くから見た，平原で座っているカピバラ)
この例では「近くから見た」という距離的な概念にも対応していることがわかります．



a ... of an eagle sitting in a field at sunrise (日の出に平原に座る... のワシ)
... にはそれぞれの画像の上に書かれている文章が入ります．一番右の a painting in the style of van gogh であれば，「日の出に平原に座るゴッホ風の作画のワシ」となります．このように文章で画風を指定することも可能になっていることがわかります．

DALL-E は画像とその対となる文章を学習させただけで，「空間情報」や「作風」といった概念を言語から視覚へと変換することが可能になったのです．

1.2 パーセプトロン

1.2.1 パーセプトロンの仕組み

パーセプトロンは 1957 年にローゼンブラッドという研究者によって開発されたプログラムです。このパーセプトロンですが、ニューラルネットワークそして多層ニューラルネットワークであるディープラーニングに繋がる大事なアルゴリズムです。パーセプトロンとはある信号を入力として受け取り、あるルールに基づいて変換し、特定の信号を出力するものです。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。ただし、ここで言う信号は情報を先へと伝達していく、ある種川の流れのようなものがありますが、パーセプトロンの信号は流す、流さないの 2 種の値です。以下では流す場合を 1、流さない場合を 0 として記述していきます。

例えば図 1.1 のような 2 つの入力 (x1,x2) と 1 つの出力 (y) の単純パーセプトロンを考えます。ここで w1 と w2 は入力に対する重みとなります。ここでいう重みとは入力の重要度を調整する値のことです。例えば重みの 1 を重視したい！となったとき w1 を 0.7、w 2 を 0.3 などとすることで調整できます。図 1.1 での w0 はバイアスです。バイアスは 1 を出力する度合いを調整するための値です。バイアスを重みという事もありますが、前述の w1,w2 のような重みとは異なります。また、図 1.1 のそれぞれの◯のことをノードまたはニューロンと呼び、出力が限界値（この場合 1）を超えるとニューロンが発火すると表現することもあります。

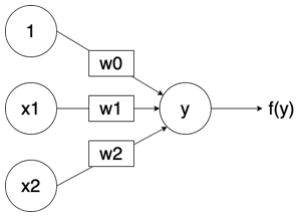


図 1.1　パーセプトロン概要 ([4] より引用)

出力 y は決められた閾値に応じて反応し、特定の閾値 θ を越えた場合 1 を出力し、越えなかった場合 0 を出力します。これを式で表すと以下のようになります。

$$y = 0(w1x1 + w2x2 + w0 \leqq \theta)$$
$$y = 1(w1x1 + w2x2 + w0 > \theta)$$

1.2.2 単純な論理回路

ここで例えば以下のような x1,x2 の組み合わせに対して、y を出力したいとします。これに関して、

x1	x2	y
0	0	0
1	0	0
0	1	0
1	1	1

w0,w1,w2, θ の組み合わせはたくさん考えられます。例えば (w0,w1,w2, θ) が (0.1,0.4,0.4,0.6) などが例に

なります。これを先ほどの式に当てはめるとy=0.4x1+0.4x2+0.1 と 0.6 の大小を考えることになります。今回は手動で行いましたが、これを機械が調整してくれるのが機械学習です。

このような上記の回路。つまり、二つの入力がどちらも 1 となる時のみ出力が 1 となる回路を AND ゲートと言います。逆に二つの入力がどちらも 1 となる時のみ出力が 0 となる回路を NAND ゲートと言います。具体的な数値でいえば (0.1,-0.5,-0.5,-0.4) の時などで成立します。

同じ流れで OR ゲートについても考えます。OR ゲートは以下のような論理回路で、入力信号の少なくとも 1 つが 1 であれば出力が 1 になる論理回路です。具体的な数値でいえば（0.2,0.5,0.5,0.3）で成立します。

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	1

以上のようにパーセプトロンを用いれば AND,OR,NAND という論理回路が表現できることがわかった。

1.2.3 パーセプトロンの限界

AND、OR に続き、XOR ゲートについても考えていきましょう。XOR ゲートとは以下のような論理回路で排他的論理和とも呼ばれます。

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0

この XOR ゲートを実装するためにはどのような重みパラメータを設定すべきでしょう。結論から述べるとこれまでのパーセプトロンでは実装することができません。実装不可であることをここでは視覚的に考えていきたいと思います。今までのパーセプトロンは前述の通り、y=w1x1+w2x2+w0 で表わしてきました。例えば OR ゲートの挙動を視覚的に考えてみると

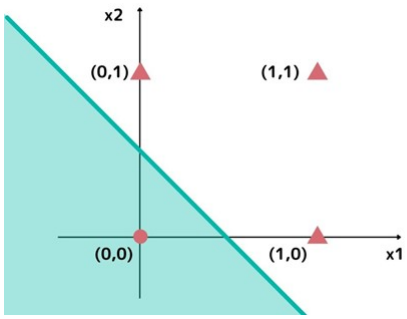


図 1.2　パーセプトロンの可視化 ([5] より引用)

○が 0、△が 1 を表すことで上記のように視覚化が可能です。OR ゲートを作るためには○と△を直線に

よって分ける必要がありますが、図??からもわかるように直線で 4 つの点を正しくわけることが可能です。それでは XOR ゲートについても同様に考えてみましょう。

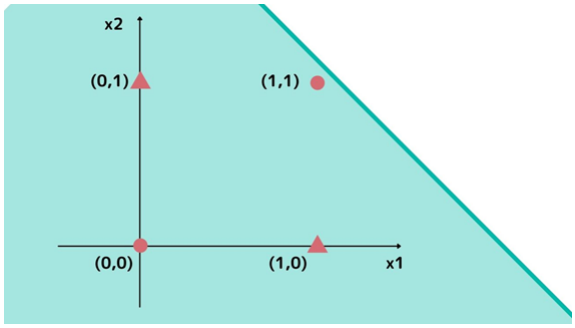


図 1.3　XOR ゲートに関するパーセプトロンの可視化 ([5] より引用)

図 1.3 からもわかる通り、○と△を直線で分けようとするのは難しいことが分かります。直線で判別することを線形判別と言いますが、XOR ゲートは線形判別ではうまく分けることができない組み合わせという事になります。しかし、このような組み合わせは現実問題に多く発生します。

1.2.4　多層パーセプトロン

ではこのような XOR ゲートを作るにはどのようにしたら良いでしょうか。一つの方法はこれまで述べた AND、NAND、OR ゲートを組み合わせて配線する方法です。

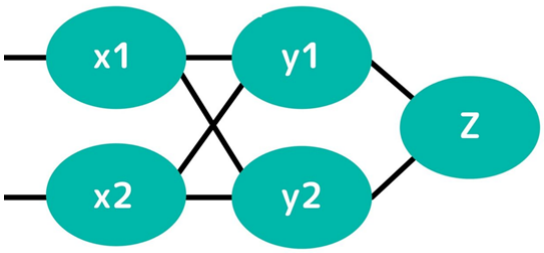


図 1.4　複数パーセプトロンによるゲートの実現 ([5] より引用)

上記のように複数パーセプトロンを組み合わせたとします。例えば x1,x2,y1,y2 の組み合わせを以下のようにすることで出力となる z をＸＯＲゲートと同様の出力にすることが可能です。

x1	x2	y1	y2	z
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

上記の組み合わせについて考えると、y1 と y2 は AND ゲートを用いることで z が出力でき、x1 と x2 は NAND ゲートを用いることで y1 を、OR ゲートを用いることで y 2 を出力することが可能です。

1.2.5 まとめ

多層パーセプトロンはこれまでみてきた回路よりもより複雑な回路の生成が可能であり、層を重ねることで非線形な表現も可能になります。そのことより、必要な部品を段階的に作り上げていくことで、理論上コンピュータが行う処理も表現することが可能です。

1.3　ニューラルネットワーク

1.3.1　パーセプトロンとの違い

ニューラルネットワークは神経細胞とそのつながりを模したもので, ニューロンのつながり方はパーセプトロンと同じである. パーセプトロンとの違いは重みやバイアスを学習して自動で定められるようしている点にある. 入力層と中間層と出力層からなるニューラルネットワークの構造の例を図 1.5 に、入力層から中間層へのつながりの一部を拡大したものを図 1.6 に示す. ただし図 1.6 ではバイアスの項を明示するために入力層のノードを一つ増やして表記した.

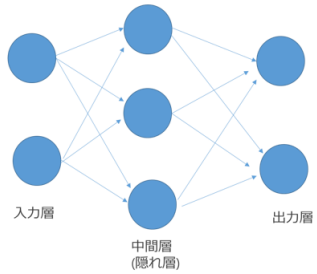


図 1.5　ニューラルネットワークの構造の例

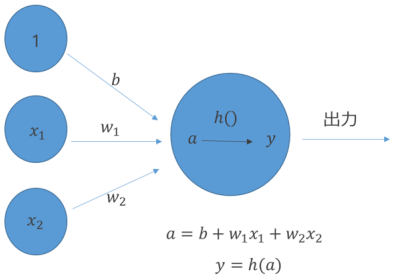


図 1.6　拡大図

パーセプトロンでは入力に重みをかけてバイアスを足したものが 0 以下なら 0 の, 0 より大きいなら 1 の信号を伝える. つまり, 入力値を x_1, x_2 それぞれの重みを w_1, w_2, バイアスを b, 中間層からの出力を y として数式で表すと

$$
\begin{aligned}
a &= b + w_1x_1 + w_2x_2 \\
y &= h(a) \\
&= \begin{cases} 0 & a \leq 0 \text{ のとき} \\ 1 & a > 0 \text{ のとき} \end{cases}
\end{aligned}
$$

となる. 関数 $h(x)$ は $x \leq 0$ で 0 を $x > 0$ で 1 を出力するステップ関数である. このステップ関数のように前の層 (入力層) からの信号の重み付き総和を次の層にどのように伝えるか (発火させるか) を決める関数を活性化関数という. パーセプトロンでは活性化関数としてステップ関数を用いているが, ニューラルネットワークでは別の関数を用いる. この活性化関数の違いがパーセプトロンとの大きな違いを生む. 続いて活性化関数について述べる.

1.3.2　中間層の活性化関数

ニューラルネットワークで用いられる活性化関数の代表的なものにジグモイド関数と ReLU 関数がある. 式で表すと, ジグモイド関数は

$$
y = h(x) = \frac{1}{1 + \exp(-x)}
$$

ReLU 関数は

$$
y = h(x) = \begin{cases} 0 & x \leq 0 \text{ のとき} \\ x & x > 0 \text{ のとき} \end{cases}
$$

となる．ステップ関数, ジグモイド関数, ReLU 関数を図 1.7 に示す．

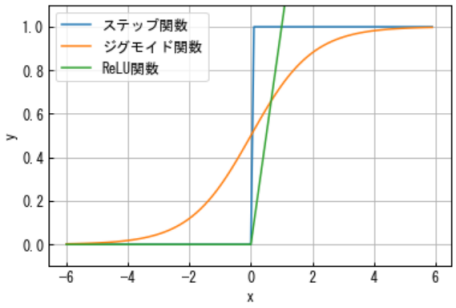


図 1.7　様々な活性化関数

ステップ関数は 0 か 1 の 2 値しか出力せず, また微分をすると x = 0 で ∞ に, $x \neq 0$ で 0 になるのに対し, ジグモイド関数と ReLU 関数は出力も, 関数の微分値も様々な値を出力することに特徴があり, これがニューラルネットワークの学習で重要な役割を果たす．(1.2.6 節の誤差逆伝搬ができるようになる.) また, 活性化関数の大きな特徴は非線形関数であることにある．線形関数であれば層を深くしても同じ値の出力を 1 層の重みで表現できてしまうからである．非線形関数にすることで表現の幅が広がり層を深くしてより精度のよいニューラルネットワークを構築できる．

1.3.3　出力層の活性化関数

機械学習で扱う問題には, 部屋の数や築年数などの入力データから不動産の価値を予測するといった連続値を出力する回帰問題と, 数字の書かれた画像を入力してその数字が 0 から 9 のいずれであるかを予測するといったどのクラスに属するかを出力する分類問題の二つがある．回帰問題では出力層に入力された値をそのまま出力する．つまり活性化関数として $y = x$ を用いる．一方で分類問題は, 出力層の出力数はクラスの数に一致しており, それぞれのクラスである確率を出力する．活性化関数としてソフトマックス関数を用いる．クラス数が n であり k $(k = 1...n)$ 番目のソフトマックス関数への入力を x_k とするとその出力 y_k は

$$y_k = h(x_k) = \frac{\exp(x_k)}{\sum_{k=1}^{n} \exp(x_k)}$$

と表せられる．総和をとると, $\sum_{k=1}^{n} y_k = 1$ であるからソフトマックス関数により確率に変換されることが分かる．

入力に重みをかけて活性化関数を通して出力する一連の計算を行う層を何層も重ねてニューラルネットワークが構築される．

1.4　損失関数

1.4.1　損失関数とは

ここではニューラルネットワークを学習させるために必要な**損失関数**について説明します．損失関数とは「ニューラルネットワークの出力が正解とどのくらい離れているか」を表す指標になります．従って損失関数の値を最小化することができれば，ニューラルネットワークは学習できた状態になると言えます．そして「関数の値を最小化する」というのは計算機でも可能なため (実際には条件がつく場合が多いと思いますが)，ニューラルネットワークの学習が可能になります．

例えば，入力された画像が犬か猫か判定するニューラルネットワークを考えることにします．このニューラルネットワークは入力画像が犬である確率と猫である確信度のそれぞれを出力します．すなわち，ニューラルネットワークの出力は 2 次元のベクトルになります．例えば $(0.7, 0.3)$ という出力であれば，7 割の確信度で犬だと判断したという解釈をします．確信度というのは例えば，誰がどう見ても犬である画像に対しては確信度が 0.99 になるでしょうし，犬と猫の間のような動物の画像を入力すると確信度は 0.5 辺りになると予想できます．図 1.8 がイメージ図です．

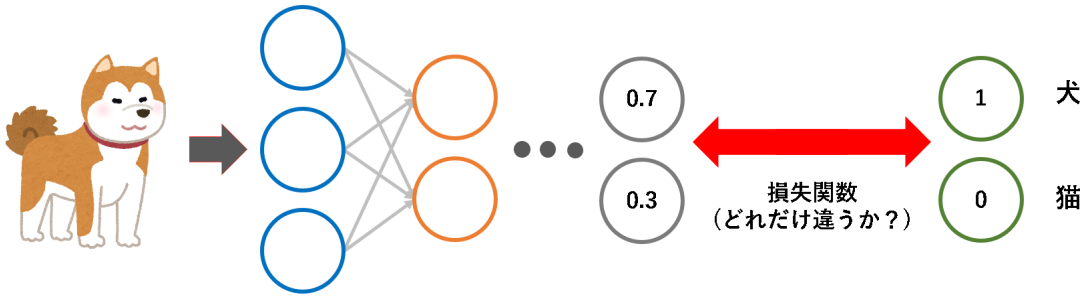


図 1.8　損失関数のイメージ

1.4.2　交差エントロピー誤差

上記の犬と猫を見分けるニューラルネットワークの場合，どんな損失関数が考えられるでしょうか？よく使われるのは**交差エントロピー誤差 (Cross Entropy Loss)** です．ニューラルネットワークの出力ベクトルを $\mathbf{y}$，正解ベクトルを $\mathbf{t}$，各ベクトルの次元を N とすると交差エントロピー誤差 $L(\mathbf{y}, \mathbf{t})$ は以下の式で表されます．

$$L(\mathbf{y}, \mathbf{t}) = -\sum_{i=1}^{N} t_i \log y_i \tag{1.1}$$

例えば犬と猫を見分けるニューラルネットワークにおいて，犬の画像を入力した時に出力が $\mathbf{y} = (0.7, 0.3)$ だったとします．犬の画像を入力した時には $(1, 0)$ と出力してくれるのが理想なので，正解ベクトルは $\mathbf{t} = (1, 0)$ となります．交差エントロピー誤差を計算すると

$$L(\mathbf{y}, \mathbf{t}) = -1 \cdot \log 0.7 - 0 \cdot \log 0.3 \approx 0.3567$$

となります．

交差エントロピー誤差はソフトマックス関数と相性が良いです．ソフトマックス関数には exp 関数が登場しますから，交差エントロピー誤差を求める際に log と打ち消しあってくれるのは想像しやすいと思います．実際，ソフトマックス関数を最終層とするニューラルネットワークの出力ベクトルを $\mathbf{y}$，正解ベクトルを $\mathbf{t}$ とすれば，交差エントロピー誤差を y_i で微分した値は $y_i - t_i$ という極めて簡単な式になります．実装に際しても，学習に必要なのはこの微分した $y_i - t_i$ という結果だけなので，ソフトマックス関数及び交差エントロピー誤差を計算せずに $y_i - t_i$ のみを計算することが多いです．このような理由から分類では交差エントロピー誤差がよく用いられます．

1.4.3 二乗和誤差

他に代表的な損失関数としてを**二乗和誤差 (Mean Squared Loss)** があります．二乗和誤差とは，出力と正解の差分を 2 乗したものの和です．ニューラルネットワークの出力ベクトルを $\mathbf{y}$，正解ベクトルを $\mathbf{t}$，各ベクトルの次元を N とすると二乗和誤差 $L(\mathbf{y}, \mathbf{t})$ は以下の式で表されます．

$$L(\mathbf{y}, \mathbf{t}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2 \tag{1.2}$$

係数の $1/2$ については，式 1.2 を微分した時に係数が 1 になり，式が簡単になるようについています．

例えば犬と猫を見分けるニューラルネットワークにおいて，犬の画像を入力した時に出力が $\mathbf{y} = (0.7, 0.3)$ だった場合，二乗和誤差を計算すると (正解ベクトルは $\mathbf{t} = (1, 0)$ ですから)

$$L(\mathbf{y}, \mathbf{t}) = \frac{1}{2}\left\{(0.7 - 1)^2 + (0.3 - 0)^2\right\} = 0.09$$

となります．

1.5 勾配法

1.5.1 どう最小値を求めるか？

例えば, $y = x^2 + 2x$ の最小値を求めてみましょう．これは x で微分した値が 0 になる点, つまり $2x + 2 = 0$ となるような x を求めればよく，簡単に $x = -1$ で最小値 -1 をとることがわかります．ではニューラルネットワークの損失関数についてはどうでしょうか？基本的にニューラルネットワークはいくつかの層からなります．またその中には活性化関数も含まれます．その場合損失関数を最小化しようとして，各パラメータについて偏微分して 0 となる点を求めて，という計算は複雑すぎて不可能になります．そこで勾配法と呼ばれる方法で，損失関数を徐々に最小に近づけていくことにします．

1.5.2 勾配降下法

勾配 (gradient) とはある関数を各変数で偏微分した値を要素に持つベクトルのことです．例えば $z = x^2 + y^2$ という関数を考えましょう．これを x, y のそれぞれで偏微分すると，$\partial z/\partial x = 2x, \partial z/\partial y = 2y$ となります．例えば点 $(1, 1)$ における勾配は $(2, 2)$ となります．これは $z = x^2 + y^2$ のグラフをイメージしてもらえるとわかると思いますが，点 $(1, 1)$ から見た最小値 (原点) の方向は $(-2, -2)$ と同じ方向となっています．このように勾配にマイナスをかけたベクトルの方向というのは関数の値が小さくなる方向ということになります．従って，損失関数の値を各パラメータで偏微分し (先の例では x, y が各パラメータの値にあたります)，勾配を求め，その逆方向に進んでいけば損失関数の値は小さくなってくれるでしょう．この方法を**勾配降下法**と呼びます．

勾配降下法はよく山で例えられます．もし山にいるとして，現在地がわからないとします．どうにかして山を下らなければなりませんが，地図もありません．しかし，今の場所の地面がどの方向に傾いているかは体感で理解できます．そこで傾いている方向に歩いていくという方法を取れば，いつかは下山できるでしょう．これが勾配降下法のやっていることです．

1.5.3 パラメータの更新

あるパラメータ w を勾配降下法で更新することを考えます．これを繰り返すことでパラメータが最適な値に近づき，損失関数の値も最小に近づいていきます．勾配降下法は勾配の逆方向へ移動していくので，パラメータ w で偏微分した値を引くことになります．式で表せば以下のようになります．ただし更新後のパラメータを $\hat{w}$ で表しています．

$$\hat{w} = w - \eta \frac{\partial L}{\partial w}$$

ここで，係数 η を使っています．これは**学習率**と呼ばれます．学習率はパラメータの更新 (学習) をどれだけのステップで行うかを表します．大きすぎると微調整が利かないためパラメータが最適に近づきにくくなります．逆に小さすぎると必要な更新の回数が増えてしまい学習に時間がかかってしまうことになります．このように学習率はちょうどいい値に設定しなければなりません．しかし学習率を求める公式のようなものは存在せず (少なくとも筆者は知りません)，いくつかの候補で学習させてその中から最も良い値を選ぶなど手間のかかる方法で求めるしかありません．

1.5.4　勾配降下法の欠点

もちろん勾配降下法には欠点があります．簡単に 1 変数で考えてみましょう．図 1.9 は $f(x) = 2x^4 - 4x^3 - 5x^2 + 10x + 9$ のグラフです．$f(x)$ の最小値は $0(x = -1)$，グラフの赤丸です．$x < 0$ のような範囲では，勾配降下法，つまりより低い場所へ移動する戦略で最小値にたどり着くことができるでしょう．しかし，$x > 1$ のような範囲では黄四角の点へたどり着いてしまいます．この点は極小値であり最小値ではないため，間違った解を得たことになります．他には黒三角のように勾配が 0 となる点にたどり着いてしまうと，パラメータを更新しても値が変わらなくなってしまいます．勾配降下法にはこのような欠点があるため，実際には他の更新式が用いられることもあります．

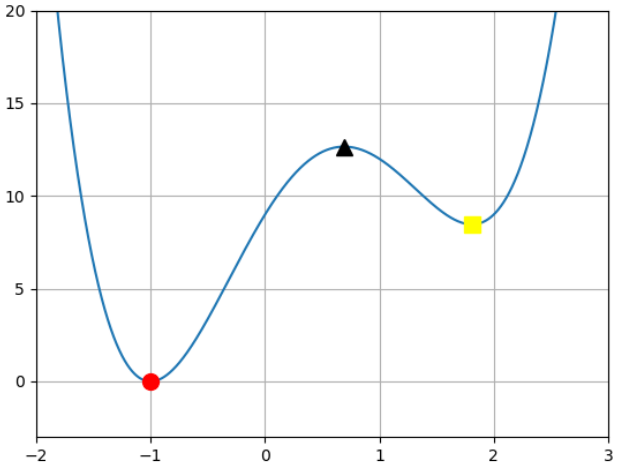


図 1.9　$f(x) = 2x^4 - 4x^3 - 5x^2 + 10x + 9$ のグラフ

1.5.5　誤差逆伝播法

しかし，やはり何千，何万，実際に使われているモデルともなると何万，何億もあるパラメータそれぞれで損失関数を偏微分した値を計算するのは簡単ではありません．そもそもコンピュータでどのように (偏) 微分を計算するのでしょうか？

ここで微分の定義式を思い出してみましょう．関数 $f(x)$ を x で微分した値 $f'(x)$ は式 1.3 で定義されます．

$$f'(x) = \lim h \to 0 \frac{f(x+h) - f(x)}{h} \tag{1.3}$$

これを素直にコンピュータで近似しようとすると，$h = 0.00001$ として $f((x+h) - f(x))/h$ を計算すればよいでしょう．これを**数値微分**と言います．もちろん，$h = 0.001$ や $h = 0.000000000001$ としても良いでしょう．しかし h の値は大きすぎると誤差も大きくなり，逆に小さすぎるとコンピュータで表現できる小ささの限界を超えてしまい，正しく計算できなくなってしまいます．また，各パラメータで偏微分となると，この計算をパラメータ数だけ繰り返さねばなりません．数値微分は精度と速度のどちらの面でも勾配降下法に使うには不向きです．

これを解決したのが**誤差逆伝播法 (backpropagation)** です．例えば 2 つの関数，$z = x^2 + y^2, w = z^2$ を考えます．$(x, y) = (0.5, 0.3)$ での勾配を求めましょう．まず，それぞれの値を計算します．

$$z = 0.5^2 + 0.3^2 = 0.34$$
$$w = 0.34^2 = 0.1156$$

ここで，w の偏微分で以下の連鎖律を使います．

$$\frac{\partial w}{\partial x} = \frac{\partial w}{\partial z}\frac{\partial z}{\partial x}$$
$$\frac{\partial w}{\partial y} = \frac{\partial w}{\partial z}\frac{\partial z}{\partial y}$$

それぞれ計算します．

$$\frac{\partial w}{\partial z} = 2z = 2 \times 0.34 = 0.68 \tag{1.4}$$
$$\frac{\partial z}{\partial x} = 2x = 2 \times 0.5 = 1 \tag{1.5}$$
$$\frac{\partial z}{\partial y} = 2y = 2 \times 0.3 = 0.6 \tag{1.6}$$

よって

$$\frac{\partial w}{\partial x} = 0.68 \times 1 = 0.68$$
$$\frac{\partial w}{\partial y} = 0.68 \times 0.6 = 0.408$$

となります．実際，$w = x^4 + 2x^2y^2 + y^4$ ですから，

$$\frac{\partial w}{\partial x} = 4x^3 + 4xy^2 = 0.68$$
$$\frac{\partial w}{\partial y} = 4y^3 + 4x^2y = 0.408$$

となり，上記の計算方法でも問題なく勾配を求めることができます．ここで重要なのが値を再利用できるという点です．まず式 1.4 の計算で，先に計算しておいた $z = 0.34$ というのを使っています．ニューラルネットワークの場合には損失関数の値を求めるために一度計算しているはずなのでその値を保存しておけばよいことになります．また，連鎖律から $\partial w/\partial z$ の部分が共通しているため，式 1.4 の計算は一度だけすればよいこともわかります．ニューラルネットワークの場合には層がいくつもありますが，連鎖律を考えるとほとんど同じ計算をすることになります．その計算を一度だけすればよいというのは計算時間の短縮においてかなり嬉しいことです．これが**誤差逆伝播法**の考え方になります．パラメータが増えれば増えるほど，数値微分と誤差逆伝播法の計算時間の差は大きく大きくなっていきます．

誤差逆伝播法はフレームワーク (実装の時に必要となる機能を簡単に使えるようにまとめたもの) では**自動微分**として用意されており，簡単に使えるようになっています．しかし，実際に自分の手で誤差逆伝播法を実装してみたいという方は [6] がおすすめです．Python と Numpy でニューラルネットワークの学習を一通り実装できる本となっています．

2　音声変換 AI(デモ紹介)

2.1　概要

今期は音声に関するモデルを実装したいという声があり，最終的に音声変換モデルを実装しました．現在，KaiRA の HP で公開していますのでぜひお試しください．

ここで言う音声変換とは，A さんの声が入力された時に，それを B さんの声に変換することを指します．ただし声質以外の情報，例えば喋っている内容だったり，アクセント，訛り，滑舌，発音の癖，などなどは変換の前後で不変でなければなりません．関西弁で喋った音声を，アメリカ大統領のスピーチの音声での声質に変換したとしても，関西弁が崩れてはいけないのです．ここではその音声変換 AI の仕組みについて説明していきたいと思います．

2.2　音声変換 AI の仕組み

今回実装した音声変換モデルは StarGANv2-VC[7] を使用しています．名前からもわかる通り，StarGANv2-VC は StarGAN[8], StarGANv2[9] をベースにしています．

まず，StarGAN と StarGANv2 について説明します．これらは元々音声変換ではなく画像変換を行うモデルでした．例えば StarGANv2 で変換した画像を図 2.1 に示します．

図 2.1　StarGANv2[9] による画像変換

図 2.1 は，上行と左列が実在人物の写真で，中央の 3 × 5 のグリッド画像が変換した画像となっています．同じ列では顔が共通しており，同じ行ではスタイルが共通しています．これを音声に応用したものが

StarGANv2-VC となっています．

もう少し技術について触れます．このような画像変換，音声変換で最大の課題となるのが「正解が存在しない」ということです．例えば「A さんの写真の髪色を B さんのものにしたい」と思ったとしても，基本的にはそのような写真は存在しません．画像加工やメイク技術で頑張って似せたとしても，それは「似せた」だけであって本質的に同じになったわけではありません．(色のムラだったりが存在するでしょう)

ここで，機械学習には「教師あり学習」と「教師なし学習」の 2 種類があります．例えば人の顔認識というのは正解があります (自撮りをすればそれは疑いようなくあなた自身の写真になります) から，その正解を使って学習すれば教師あり学習に分類されます．それに対して今回のように正解のないものを学習するのは教師なし学習に分類されます．当然正解がない分，上手く工夫をしないと学習は不可能です．

では StarGAN シリーズはどのように学習させているのかというと，主となるのは **Cycle Consistency Loss** というものです (実際には他にも複数の損失関数を組み合わせて学習させていますが)．概念図を 2.2 に示します．同じ色は同じ声質であることを表します．これは，「A さんの声を B さんの声に変換した声を再び A さんの声に変換すると，元の声に戻るはずだ」というやり方です．この方法では「変換する前の A さんの声」という正解が存在することになり，学習が進むようになります．

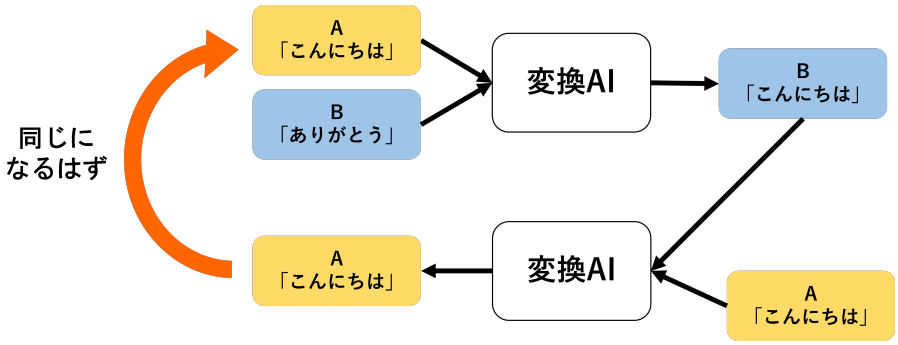


図 2.2　Cycle Consistency Loss の概念図

2.2.1　工夫点

ここからは今回の実装に関して工夫した点を紹介します．なお実装は [10] からフォークしたものを使っています．変更を加えたのは以下の 2 点です．

1. Domain 間のパラメータの共有
2. 学習データの増強

Domain 間のパラメータの共有

そもそも StarGANv2-VC は変換先の声質が固定されており，任意の声に変換するものではありませんでした (所謂 Any-to-Many)．その変換先の声質を Domain と呼んでいます．例えば 10 人分の音声データで学習させれば Domain は 10 個あることになります．この理由は後述する Adversarial Classifier Loss に関係します．そこで汎用的に任意の声に変換できるように，モデルのパラメータの一部が Domain 間で別々になっていたのを共用としました．

学習データの増強

先述の通り元々は StarGANv2-VC は固定の声質へ変換するモデルだったため 10 人分規模のデータで学

習させていました．これを 100 人規模のデータを使用するよう変更しました．なお学習データには JVS corpus[11] を使用しました．これは声優や俳優の方が日本語で録音した音声を 100 人分集めたデータです．

これらの工夫により任意の声質に変換できるモデル (所謂 Any-to-Any) に近づきました．

2.2.2 補足

GAN について知見がある方向けの補足です．StarGANv2-VC では変換の品質を向上させるため，**Adversarial Classifier Loss** を提案しています．概要図を図 2.3 に示します．ここでの Domain は学習データに含まれる音声の話者と考えてもらって結構です．Discriminator は Generator がどの Domain に変換したとしても，その変換元の Domainy_k を判定できるように学習します．対して Generator はどの Domain から変換したとしても Discriminator が変換先の Domainy_k だと (変換元の Domain がわからないように) 判定するように学習します．

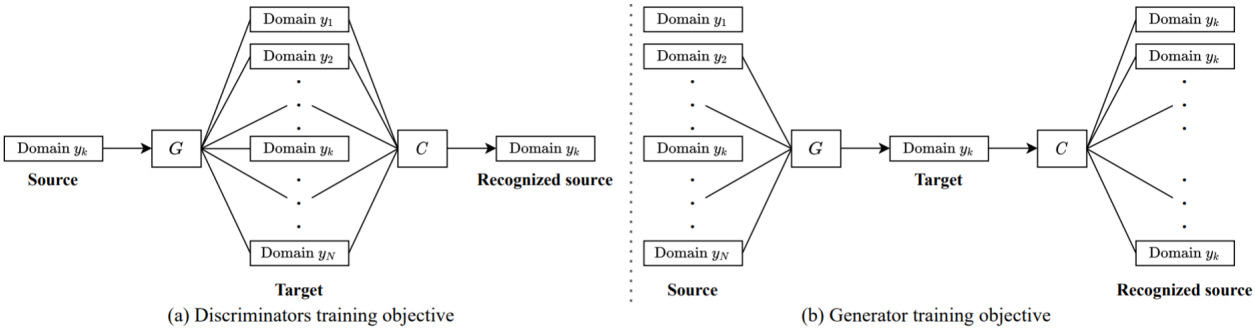


図 2.3　StarGANv2-VC[7] で提案された Adversarial Classifier Loss の概要図

Adversarial Classifier Loss を使用すること自体が今回の Any-to-Any モデルの構築という目標に反するかもしれませんが，StarGANv2-VC の変換に汎用性を持たせることで Any-to-Any に近づけることができました．

3　Spiking Neural Network の紹介

3.1　はじめに

Deep Learning 技術は近年十分に成長してきているがその目標とは一体どのようなものなのであろうか．もちろん複雑なタスクの達成が挙げられるが，他にも思い浮かべられるものの一つとして周辺機器，つまりエッジデバイスでの Deep Learning による推論が挙げられる．エッジデバイスで推論ができるようになれば，それこそ名探偵コナンに出てくる蝶ネクタイ型変声機や映画アイアンマンのスーツの処理系統のようなものが出来るようになるかもしれない．しかし現在の Deep Neural Network(DNN) では精度はあるが消費電力がどうしてもボトルネックとなり実現が難しい．そこで第三世代ニューラルネットワークとも言われる Spiking Neural Network(SNN) が登場する．Float 型で計算を行っていた DNN に対して，スパイク (0 か 1 かの 2 値のパルス) に限定するため処理が少なく低消費電力が実現される．この性質については専用のニューロモーフィックチップを用いた検証がされており，実際にアプリケーションとして世間に広まるのも近いと考えられる．

この章では SNN の成り立ちとその動作についての説明を簡単にし，次に SNN を用いたアプリケーション例を紹介する．最後にまとめと SNN 関連でのおすすめの文献を紹介して締めたいと思う．

3.2　SNN の成り立ち

前節では消費電力が低いニューラルネットワークとして SNN を紹介したが，もともとは低消費電力を目指して開発されたものではない．”Spiking”とあるように脳の情報伝達で使われるスパイクをそのままニューラ

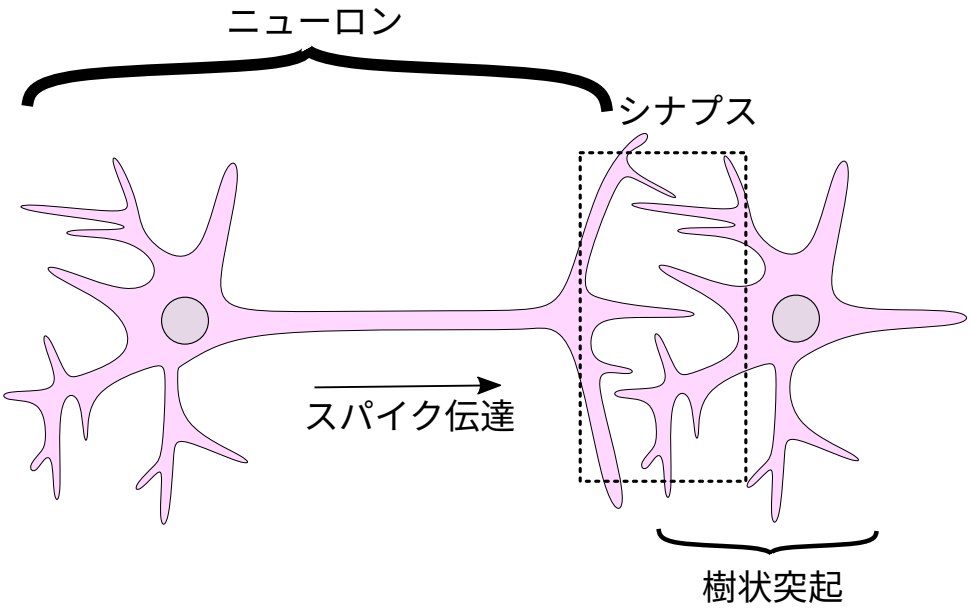


図 3.1　ニューロンとスパイク伝達の概要図

ルネットワークで用いることで新しいニューラルネットワークのパラダイムをなすことができるのではないか

という考えがあった．

実際の脳では図 3.1 のように，生体の神経ネットワークではニューロンが基本単位となっている．スパイクはニューロンを伝わっていくが，隣り合うニューロンと直接接続されているわけではなくシナプスと呼ばれる構造の隙間が間にあるためにそのままスパイクが流れることはない．その代わりシナプスでは神経伝達物質を媒介にしてスパイクを伝達している．この際にニューロン間のつながりの強さを意味する「重み」がスパイクにかけられる．この重みやスパイクの発火頻度を実数として数理的にモデル化することでニューラルネットワークの前進となるパーセプトロンが開発され，そのパーセプトロンを多層にすることによりニューラルネットワークが生まれた．

それに対して，1997 年にニューラルネットワークに実数ではなくスパイクを入力として計算させたモデルの提案と検証を行われた [12]．この研究で SNN の一般的な演算能力が認められ，第三世代ニューラルネットワークといわれるようになった．

3.3 SNN の動作

この説では SNN の動作について DNN の動作と比較して説明する．これ以降は DNN を Artificial Neural Netowork(ANN) と呼ぶ．ANN は一般的にはニューラルネットワーク全般を指すが, SNN という分野ではスパイクを用いる SNN と比較して ANN と呼ばれる．

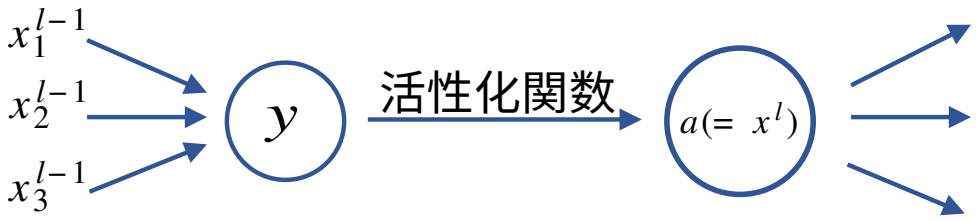


図 3.2　l 層の ANN の順方向の動作

ANN は図 3.2 のように実数の入力 x_1^{l-1},x_2^{l-1},x_3^{l-1} を重み w とバイアス b により線形変換を行い y を求め，活性化関数 $h(y)$ を用いて非線形変換を行う．この活性化値 a が次層への入力値 x^l になる．これらを式で表すと式 3.1, 式 3.2 のようになる．

$$y = \sum_i w_i x_i^{l-1} + b \tag{3.1}$$

$$a = x^l = h(y) \tag{3.2}$$

この繰り返しにより特徴量抽出とともに識別則も成している．

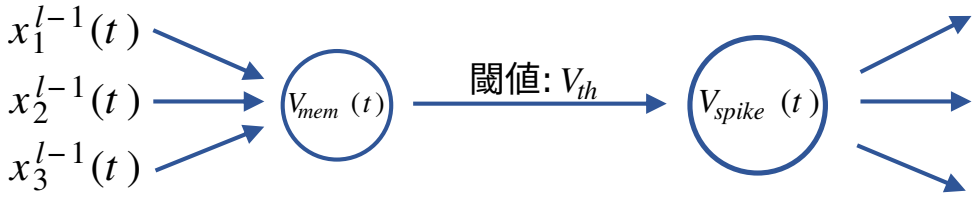


図 3.3　l 層の SNN の順方向の動作

SNN に関しても基本的には ANN と同じである．図 3.3 のように SNN では情報伝達が実数で行われるので

はなく，時系列的スパイク列 $x_i^{l-1}(t)$ の発火頻度 R_{fire} となっている．発火頻度 R_{fire} は総時間タイムステップ T, 総発火回数を C_{spike} としたとき式 (3.3) のように定義される．

$$R_{fire} = \frac{C_{spike}}{T} \tag{3.3}$$

スパイクは 0,1 の二値で表現され，ANN と同様に重み $\tilde{w}$ とバイアス $\tilde{b}$ により線形変換される．そしてその値が膜電位 $V_{mem}(t)$ に加算され，膜電位が閾値 V_{th} を超えると発火しスパイク $V_{spike}(t)$ を次層に伝える．これらを式で表すと式 (3.4), 式 (3.5), 式 (3.6) のようになる．

$$V_{mem}(t+1) = V_{mem}(t) + V_{th}\left(\sum_i \tilde{W}_i x_i^{l-1}(t) + \tilde{b}\right) \tag{3.4}$$

$$V_{spike}(t+1) = \Theta(V_{mem}(t+1) - V_{th}) \tag{3.5}$$

$$\Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else.} \end{cases} \tag{3.6}$$

以上が SNN の動作の説明である．SNN ではスパイクのみで情報伝達を行い，そのスパイク列はスパース (疎) である場合が多いため低消費電力が実現できる．

次に SNN の学習について説明する．SNN はニューラルネットワークであるので，学習する必要がある．その方法の一つに SuperSpike 法 [13] がある．この方法は教師あり学習で使われ，誤差伝搬法の近似を行うことで学習を可能にしている．また他にも STDP 則を用いた教師なし学習がある．これは図 3.4 のように前のニューロンの発火によって接続されているニューロンが発火したとき，その間のシナプスの結合が強くなる．つまりその間の重みが増える．このようにして特徴抽出のためのパラメータ更新が行われている．

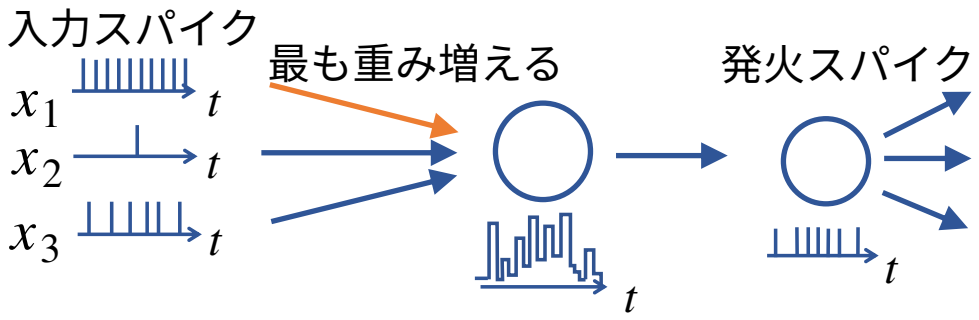


図 3.4　STDP 則による重み更新の概要図

3.4　SNN のアプリケーション

この節のタイトルとしてアプリケーションと大体的に取り上げようとしているが，その実 SNN のアプリケーション例は少ない．これについての大きな原因として SNN の学習が難しいという点が挙げられる．前節では SuperSpikie 法や STDP 則といった学習方法を説明したが，このような学習方法ではベンチマークデータセットでのクラス分類といった単純なタスクのみでしか検証されていない．

では，どのようにすれば良いのか？その答えの一つとして学習した DNN のパラメータを SNN のパラメータとして用いる，ANN-SNN 変換という方法である．この ANN-SNN 変換では ANN の活性化値と SNN の

スパイク発火頻度が対応するようにモデルパラメータを線形変換 (正規化) する．精度の高い DNN を利用することで精度の高い SNN を実現できるということが可能になるのである．

この方法を用いたアプリケーション例として Spiking-YOLO[14] を紹介する．YOLO というのは物体検知の分野で活躍するモデルである．物体検知とは，画像中のターゲットとなる物体をバウンディングボックスとよばれる四角形の領域で推定するタスクのことである．この研究では畳み込みニューラルネットワーク (CNN) に関してパラメータをチャネルごとに変換のスケールを変えることで物体検知を可能にしている．図 3.5 のように Kim らの提案手法では早く精度の高い推定が可能となっている．

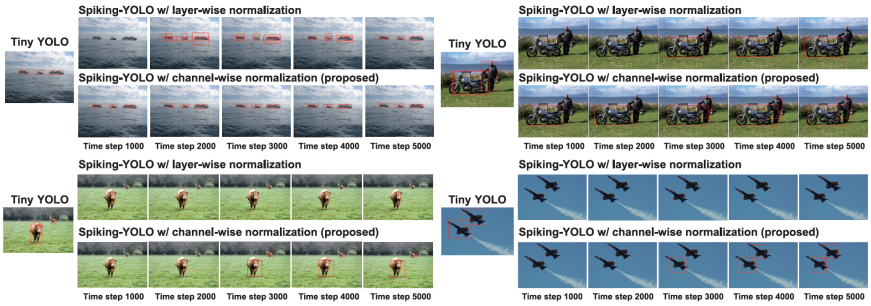


図 3.5　Tiny YOLO(DNN) による推定，既存手法 (layer-wise) と Kim らの提案手法 (channel-wise) の Spiking-YOLO による推定結果 [14]: 画像中に赤く四角い領域が YOLO が推定した領域となっている．

3.5　まとめ

スパイクを用いることで SNN は DNN よりも生物的に脳をより模倣しているモデルであり，低消費電力に優れている．現在は学習の難しさから実世界への応用は難しいが DNN から直接変換することで複雑なモデルでも精度が高い SNN モデルを実現することができている．将来，SNN 専用のニューロモーフィックチップなどがもっと発達すれば一般的に用いられるかもしれない．

3.6　おすすめ文献

レビュー論文全般

レビュー論文はその分野の背景や既存研究を丁寧にまとめてもらっている論文で，初めてその分野を研究する人やアウトラインを知りたい人におすすめの論文．

- Spiking Neural Network 技術の現状と課題に関する考察 [15]
- スパイキングニューラルネットワークにおける深層学習 [16]
- Deep learning in spiking neural networks[17]

書籍

工学徒のための脳科学についての書籍．分かりやすく面白かったです．

- メカ屋のための脳科学入門 脳をリバースエンジニアリングする [18]

同人誌ですが，SNN の実装やニューロンモデルなど数理モデルが明記されて，とても分かりやすいです．最近，web ページになってます．僕はこの本で SNN の存在を知りました．

- ゼロから学ぶスパイキングニューラルネットワーク [19]

参考文献

[1] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[2] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[3] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Dall e: Creating images from text. `https://openai.com/blog/dall-e/`, 2021-01-05. Accessed on 2021-11-18.

[4] 単純パーセプトロン. `https://mukai-lab.info/pages/classes/advanced_studies_seminar_1/chapter2/`. Accessed on 2021-11-17.

[5] ウマたん. 【これだけ！】パーセプトロンを簡単な例で分かりやすく解説！`https://toukei-lab.com/perceptron`. Accessed on 2021-11-17.

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

[7] Yinghao Aaron Li, Ali Zare, and Nima Mesgarani. Starganv2-vc: A diverse, unsupervised, non-parallel framework for natural-sounding voice conversion. *arXiv preprint arXiv:2107.10394*, 2021.

[8] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8789–8797, 2018.

[9] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8188–8197, 2020.

[10] yl4579. Starganv2-vc. `https://github.com/yl4579/StarGANv2-VC`, 2021. Accessed on 2021-08-27.

[11] Shinnosuke Takamichi, Kentaro Mitsui, Yuki Saito, Tomoki Koriyama, Naoko Tanji, and Hiroshi Saruwatari. Jvs corpus: free japanese multi-speaker voice corpus. *arXiv preprint arXiv:1908.06248*, 2019.

[12] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, Vol. 10, No. 9, pp. 1659–1671, 1997.

[13] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, Vol. 30, No. 6, pp. 1514–1541, 2018.

[14] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, pp. 11270–11277, 2020.

[15] 岡島義憲. Spiking neural network 技術の現状と課題に関する考察. 人工知能学会第二種研究会資料, Vol. 2020, No. AGI-015, p. 08, 2020.

[16] 酒見悠介, 森野佳生. スパイキングニューラルネットワークにおける深層学習. 生産研究, Vol. 71, No. 2, pp. 159–167, 2019.

[17] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, Vol. 111, pp. 47–63, 2019.

[18] 高橋宏知. メカ屋のための脳科学入門 脳をリバースエンジニアリングする. 日刊工業新聞社, 2016.

[19] HiroshiARAKI. ゼロから学ぶスパイキングニューラルネットワーク. https://booth.pm/ja/items/1585421, https://snn.hirlab.net/, https://github.com/takyamamoto/SNN-from-scratch-with-Python.

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 2 回生)
活動日　毎週木曜日 18:30〜　活動内容に応じて他曜日にも活動がある場合があります
場所　オンライン (対面可能になれば京大病院内)
会費　無料
入会資格　大学，学部，年齢は問いません
mail　kyoto.kaira@gmail.com
twitter　@kyoto_kaira

活動内容

1. **輪読会**
   Deep Learning/機械学習関連の本の輪読を章ごとに発表者を決めて行っています．扱う本は前期，後期にそれぞれアンケートを取って決めます．
2. **実装開発**
   メンバーが興味のあるテーマについて実装開発をしています．今年度から Google Cloud Platform の利用を開始したため，計算資源の提供も可能です．
3. **論文読み会**
   Deep Learning/機械学習関連の論文の読み会を行います．また最新論文をキャッチアップできるシステムも導入予定です．

文責

1. Deep Learning **入門**　三宅大貴，原田紘太郎，樋上彩加
2. **音声変換** AI(**デモ紹介**)　三宅大貴
3. Spiking Neural Network **の紹介**　羽原丈博