深層学習 day2

秋葉洋哉

2024 年 7 月 3 日

1　ニューラルネットワーク

1.1　勾配消失問題

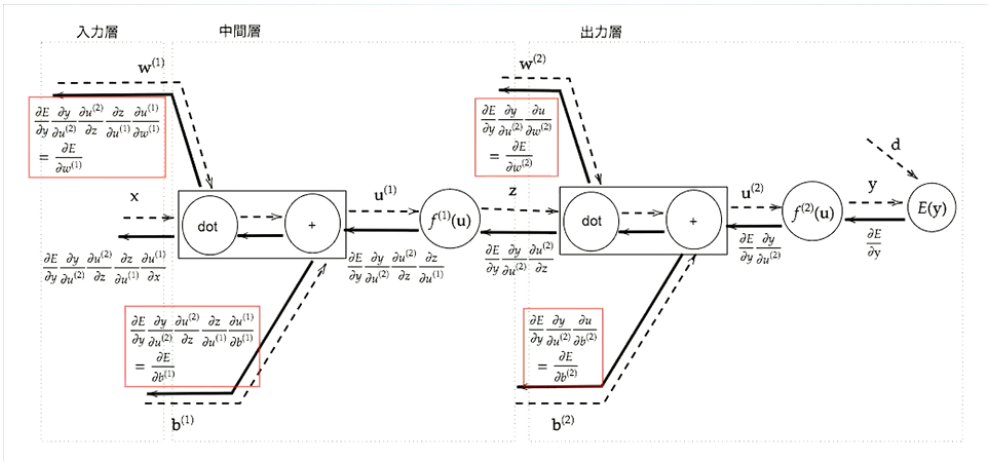


図 1: 誤差逆伝播法における勾配

勾配消失問題とは、多層ニューラルネットワークにおいて、勾配が逆伝播する際に、勾配が小さくなり、学習が進まなくなる問題である。勾配消失問題は、シグモイド関数やハイパボリックタンジェント関数を活性化関数として用いた場合に発生しやすい。シグモイド関数の微分は、

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x)) \tag{1}$$

であり、これをグラフにすると、最大値が 0.25 であることがわかる。図 1 において、活性化関数 $f(u)$ をシグモイド関数として考えた時、u の更新式である、

$$u^{(3)} \leftarrow u - \epsilon \frac{\partial E}{\partial y}\frac{\partial y}{\partial u} \tag{2}$$

$$u^{(2)} \leftarrow u - \epsilon \frac{\partial E}{\partial y}\frac{\partial y}{\partial u^{(3)}}\frac{\partial u^{(3)}}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial u^{(2)}} \tag{3}$$

$$u^{(1)} \leftarrow u - \epsilon \frac{\partial E}{\partial y}\frac{\partial y}{\partial u^{(3)}}\frac{\partial u^{(3)}}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial u^{(2)}}\frac{\partial u^{(2)}}{\partial z^{(1)}}\frac{\partial z^{(1)}}{\partial u^{(1)}} \tag{4}$$

における、$\frac{\partial y}{\partial u}, \frac{\partial z^{(2)}}{\partial u^{(2)}}, \frac{\partial z^{(1)}}{\partial u^{(1)}}$ が最大値 0.25 の少数となる。模式図では中間層が 1 層のみであるが、実際にはさらに多くの層を持つため、これらの値が掛け合わされることで、勾配が指数関数的に小さくなり、勾配消失問題が発生する。

勾配消失問題に対しては以下の 3 つの解決方法が考えられる。

1. 活性化関数の選択
2. 重みの初期値の設定
3. バッチ正規化

1.1.1 活性化関数の選択

活性化関数の選択については、ReLU 関数や Leaky ReLU 関数を用いることで、勾配消失問題を回避することができる。Leaky ReLU 関数は以下の数式で定義される。

$$f(x) = \begin{cases} x & (x > 0) \\ ax & (x \leq 0) \end{cases} \tag{5}$$

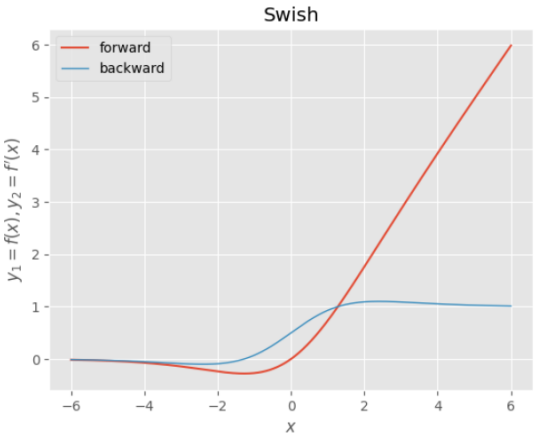
ここで、a は 0 より大きい小さな値である。Leaky ReLU 関数は、$x \leq 0$ の時に、勾配が 0 にならず、x に比例して勾配が変化するため、勾配消失問題を回避することができる。ReLU 関数は、微分値が 1 になるため先ほどのシグモイド関数における問題が解消される (図 2b)。

最近では、Swish 関数といった活性化関数も提案されている。Swish 関数は以下の数式で定義される。

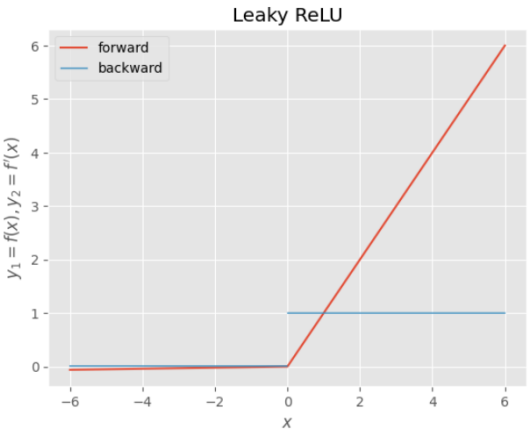
$$f(x) = x \cdot \sigma(\beta x) \tag{6}$$

$$\frac{\partial}{\partial x} f(x) = \beta f(x) + \sigma(\beta x)(1 - \beta f(x)) \tag{7}$$

ここで、σ はシグモイド関数であり、β はハイパーパラメータである。Swish 関数は、ReLU 関数を滑らかにしたものであり、ReLU 関数よりも高い性能を発揮することが報告されている (図 2a)。



(a) Swish 関数のグラフ



(b) LeakyReLU 関数のグラフ

図 2

1.1.2 重みの初期値設定

重みの初期値の設定については、活性化関数が sigmoid や tanh の場合 Xavier の初期値、活性化関数が ReLU の場合 He の初期値を用いることで、勾配消失を抑制することができる。

Xavier の初期値は、前の層のノード数を n とした時、標準偏差が $\frac{1}{\sqrt{n}}$ の分布を用いて、対象の層と前の層の間の重みを初期化する方法である。

He の初期値は、前の層のノード数を n とした時、標準偏差が $\frac{2}{\sqrt{n}}$ の分布を用いて、対象の層と前の層の間の重みを初期化する方法である。

1.1.3 バッチ正規化

バッチ正規化は、最初の入力データの正規化だけでなく、各層の出力に対しても正規化することで、勾配消失問題を回避して汎化性能を上げる方法である (図 3)。バッチ正規化は、学習時と推論時で挙動が異なるため、学習時には、ミニバッチごとに正規化を行い、推論時には、全データを用いて正規化を行う必要がある。バッチ正規化を行う際は、まずミニバッチの平均・分散を求める。

$$\mu_i = \frac{1}{m}\sum_{i=1}^{m} x_i \ (\text{平均}) \tag{8}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_i)^2 \ (\text{分散}) \tag{9}$$

次に、正規化を行う。この時、ϵ を加えることで、分母が 0 になるのを防ぐ。

$$\hat{x}_i = \frac{(\text{各値の平均からの差})}{(\text{標準偏差})} \tag{10}$$

$$= \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{11}$$

最後に、スケーリングとシフトを行う。

$$y_i = \gamma\hat{x}_i + \beta \tag{12}$$

ここで、γ はスケーリング係数、β はシフト係数、ϵ は微小値である。

正規化の種類

バッチ正規化には、他にも以下のような正規化手法がある。

- レイヤー正規化 (図 4)
- インスタンス正規化 (図 5)
- グループ正規化 (図 6)

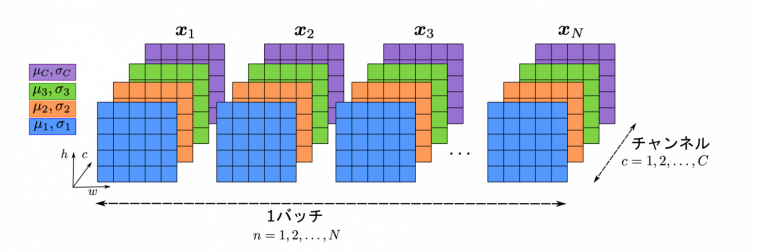


図 3: バッチ正規化のイメージ: バッチ内全体で、チャンネル単位の正規化を行う。(https://cvml-expertguide.net/terms/dl/layers/batch-normalization-layer/ より引用)

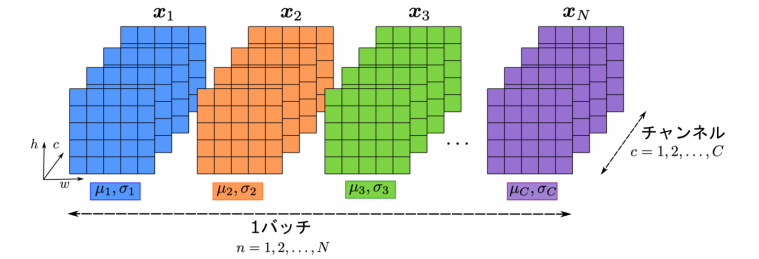


図 4: レイヤー正規化のイメージ: チャンネル内全体で、層単位の正規化を行う。(https://cvml-expertguide.net/terms/dl/layers/batch-normalization-layer/ より引用)

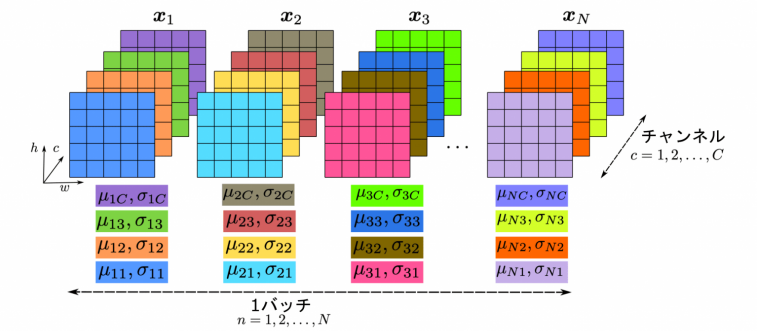


図 5: インスタンス正規化のイメージ: 画像一枚単位で、チャンネル毎の正規化を行う。(https://cvml-expertguide.net/terms/dl/layers/batch-normalization-layer/ より引用)

確認テスト

Q: 重みの初期値を 0 に設定すると、どのような問題が発生するか。簡潔に説明せよ。

A: 順伝播での出力がすべて同じ値となり、逆伝搬での勾配もすべて同じ値となるため、重みの更新がうまく行われず、学習が進まなくなる。

Q: 一般的に考えられるバッチ正規化の効果を 2 点挙げよ。

A: 学習データとテストデータの分布が異なる場合に、評価値を安定させる効果がある (汎化性能が上昇する)。また、勾配消失を解消して学習を早く進めることができる。

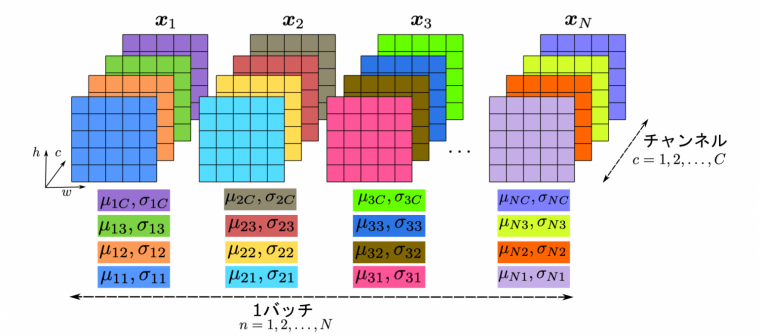


図 6: グループ正規化のイメージ: 各 1 サンプル内のグループ単位で正規化を行う。(https://cvml-expertguide.net/terms/dl/layers/batch-normalization-layer/ より引用)

1.2 学習率最適化手法

学習率最適化手法とは、学習率を適切に設定するための手法である。学習率とは、重みの更新量を調整するためのハイパーパラメータであり、大きすぎると発散し、小さすぎると収束が遅くなるため、適切な値を設定する必要がある。学習率最適化手法には、以下のような手法がある。

1. Momentum
2. AdaGrad
3. RMSProp
4. Adam

1.2.1 Momentum

Momentum は、勾配方向に加えて、前回の重みの更新量を考慮して、重みを更新する手法である。Momentum は、以下の式で表される。

$$v^{(t+1)} = \epsilon \frac{\partial E}{\partial w} - \mu v^{(t)} \tag{13}$$

$$w^{(t+1)} = w^{(t)} - v^{(t+1)} \tag{14}$$

ただし、ϵ は学習率、μ はモメンタムと呼ばれるハイパーパラメータである。前回の重みの更新量に $\mu = 0.5 \sim 0.9$ の値を掛け合わせたもの v を更新量として用いることで、勾配降下法における振動を抑える。よって、大域的最適解に向かって、谷の方向に沿って谷底を効率よく探索して収束していく特徴がある。

1.2.2 AdaGrad(Adaptive Gradient Algorithm)

AdaGrad は、学習率を重みの要素ごとに変化させる手法である。これまでの勾配の二乗和を経験としてすべて保持し、その経験に基づいて学習率を調整する。AdaGrad は、以下の式で表される。

$$h^{(t+1)} = h^{(t)} + \left(\frac{\partial E}{\partial w}\right)^2 \tag{15}$$

$$w^{(t+1)} = w^{(t)} - \frac{\epsilon}{\sqrt{h^{(t+1)}}}\frac{\partial E}{\partial w} \tag{16}$$

ただし、h はこれまでの勾配の二乗和を保持する変数である。AdaGrad は、学習率が小さくなりすぎるという問題がある。確かに、誤差関数の勾配が大きい場合、2 つ目の式で、$\frac{1}{\sqrt{h^{(t+1)}}}$ によって更新量が小さくなることが分かる。これを改善したものが RMSProp である。

1.2.3 RMSProp

RMSProp は、AdaGrad の学習率が小さくなりすぎる問題を解決するために、指数移動平均を用いて、過去の勾配の影響を減衰させる手法である。つまり、過去の経験を忘れることで、学習率が小さくなりすぎる問題を解決するということである。RMSProp は、以下の式で表される。

$$h^{(t+1)} = \alpha h^{(t)} + (1-\alpha)\left(\frac{\partial E}{\partial w}\right)^2 \tag{17}$$

$$w^{(t+1)} = w^{(t)} - \frac{\epsilon}{\sqrt{h^{(t+1)}}}\frac{\partial E}{\partial w} \tag{18}$$

ここで、α は減衰率を表すハイパーパラメータである。RMSProp は、AdaGrad の学習率が小さくなりすぎる問題を解決するが、逆に学習率が大きくなりすぎる問題がある。Adam はこれを解決するために提案された。

1.2.4 Adam

Adam は、Momentum の過去の勾配の指数関数的減衰平均と、RMSProp の二乗和の移動平均を組み合わせた手法である。Adam は、以下の式で表される。

$$m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1)\frac{\partial E}{\partial w} \tag{19}$$

$$h^{(t+1)} = \beta_2 h^{(t)} + (1-\beta_2)\left(\frac{\partial E}{\partial w}\right)^2 \tag{20}$$

$$\hat{m} = \frac{m^{(t+1)}}{1-\beta_1^{t+1}} \tag{21}$$

$$\hat{h} = \frac{h^{(t+1)}}{1-\beta_2^{t+1}} \tag{22}$$

$$w^{(t+1)} = w^{(t)} - \frac{\epsilon}{\sqrt{\hat{h}}+\epsilon}\hat{m} \tag{23}$$

ここで、β_1, β_2 はそれぞれ Momentum と RMSProp の減衰率を表すハイパーパラメータである。式 (21) と式 (22) は、バイアス補正を行うための式で、β の肩の数字は乗算を表す。

1.3 過学習

過学習とは、学習データに対しては良い性能を示すが、テストデータに対しては性能が低くなる現象である。深層学習モデルにおいて過学習が発生する原因は、以下の 3 つが挙げられる。

- パラメータ数が多い
- データが少ない
- ノード数が多い
- → ネットワークの自由度が高いということ

過学習では、重みの行列の中で、極端に大きな値を持つものが現れてしまうことによって発生する。過学習を防ぐためには、以下のような手法がある。

1. 正則化
2. ドロップアウト
3. 早期終了
4. データ拡張

1.3.1 正則化

正則化は、過学習を防ぐために、誤差関数に正則化項を追加する手法である。一般化された L(N) 正則化項は、以下の式で表される。

$$E(\mathbf{w}) + \frac{1}{p}\lambda||\mathbf{w}||_p \tag{24}$$

$$||\mathbf{w}||_p = \left(\sum_{i=1}^{n}|w_i|^p\right)^{\frac{1}{p}} \tag{25}$$

ここで、λ は正則化項の重みを表すハイパーパラメータである。L1 正則化は、$p = 1$ の場合であり、L2 正則化は、$p = 2$ の場合である。例えば、x と y の 2 次元のデータがある場合、L1 正則化は、

$$|x| + |y| \tag{26}$$

となり、L2 正則化は、

$$\sqrt{x^2 + y^2} \tag{27}$$

となる。L1 正則化を用いた回帰分析のことを Lasso 回帰と呼ぶ。Lasso 回帰は、L1 正則化を用いることで、スパースな解を得られやすいため、必要な特徴量のみを選択することができる (図 8a) 。L2 正則化を用いた回帰分析のことを Ridge 回帰と呼ぶ。Ridge 回帰は、L2 正則化を用いることで、特徴量の重みを 0 に近づけることができるため、特徴量の重みを抑制することができる (図 8b) 。

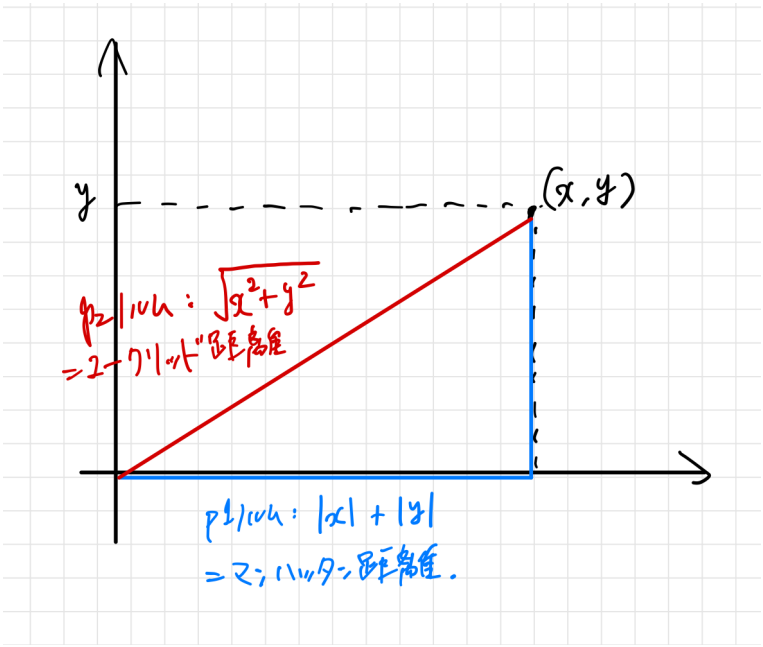
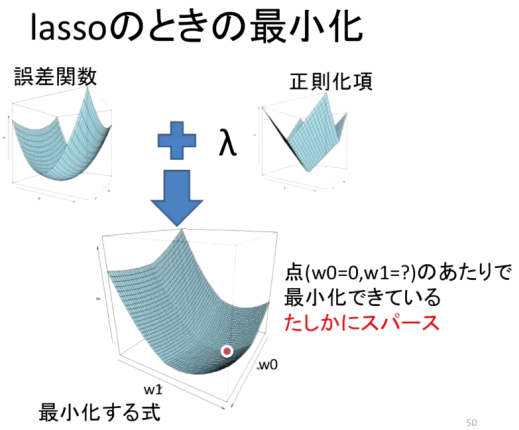
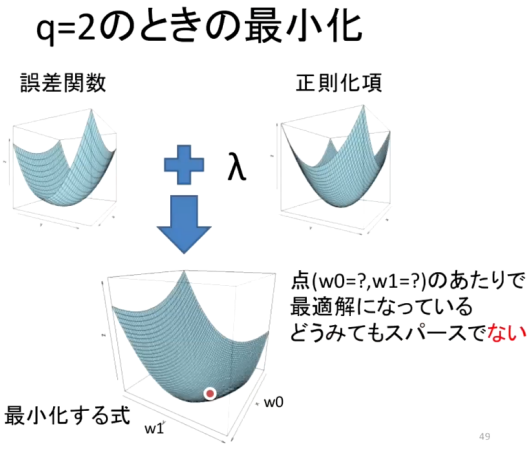


図 7: L1, L2 ノルムのイメージ



(a) Lasso 回帰における正則化のイメージ (参照: 授業資料 (Day2 18 過学習 正則化手法 4))



(b) Ridge 回帰における正則化のイメージ (参照: 授業資料 Day2 18 過学習 正則化手法 4))

図 8

Python コード

それぞれの正則化手法を用いた場合の重みの更新量は以下の通りである。

```
  # L2 正則化
    for idx in range(1, hidden_layer_num + 1):
      # 重みの勾配 ([Afiine + str(idx)].dW) に、正則化強度 (weight_decay_lambda)*重み
([params + str(idx)]) を加算する
      grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda * networ
      # 重みの更新量 (network.params['W' + str(idx)]) = 学習率 (learning_rate) * 重み
の勾配 (grad['W' + str(idx)])
      network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]

      # バイアスに正則化は適用しない
      grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
```

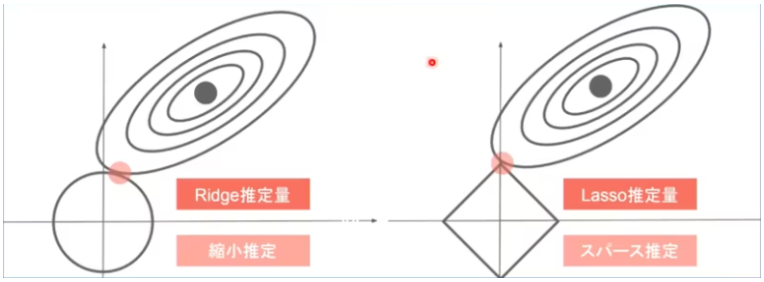


図 9

確認テスト

Q: 図 9 について、L1 正則化を表しているグラフはどちらか答えよ。
A: 順伝播での出力がすべて同じ値となり、逆伝搬での勾配もすべて同じ値となるため、重みの更新がうまく行われず、学習が進まなくなる。

1.3.2 ドロップアウト

ドロップアウトは、ニューラルネットワークの学習時に、ランダムにノードを削除することで、過学習を防ぐ手法である。ドロップアウトは、以下の式で表される。

$$y_i = r_i \times x_i \tag{28}$$

ここで、r_i は、ノード i を削除する確率を表すハイパーパラメータである。ドロップアウトは、学習時にのみ適用し、テスト時には適用しない。ドロップアウトは、モデルの自由度を下げることで、過学習を防ぐ効果がある。

2　畳み込みニューラルネットワーク (CNN)

2.1　概要

CNN は、画像認識や音声認識などの分野で高い性能を発揮するニューラルネットワークである。ある画像に対しては、隣り合うピクセルの情報が似通う可能性が高い。また、ある動画に対しては、連続するフレームの画像情報が似通う可能性が高い。そのような次元間で連続した情報を扱う際に、CNN は高い効果を発揮する。

2.2　LeNet

LeNet は、畳み込みニューラルネットワークの元祖とも言われるモデルである。CNN を理解するために、まずは LeNet の構造について説明する。LeNet における入力層は、32x32 の画像、つまり 1024 の数字の集まりであり、畳み込み層を通過することで、画像の特徴を 10 種類に分類、つまり 10 の数字に絞り込む。LeNet では、特徴マップである畳み込み層と、その要約としてのプーリング層を交互に繰り返すことで、画像の特徴を抽出している。その過程は以下の通りである。図 10 は、LeNet の概要を示している。

1. 入力層: 32x32 の画像
2. C1: 畳み込み層 (28×28×6 通りの特徴マップ) を生成
3. S2: プーリング層 (14×14×6 通りの特徴マップ) に要約
4. C3: 6 つの特徴マップを基に畳み込み層 (10×10×16 通りの特徴マップ) を生成
5. S4: プーリング層 (5×5×16 通りの特徴マップ) に要約
6. C5: 全結合層 (120 通りの特徴マップ) を生成
7. F6: 全結合層 (84 通りの特徴マップ) を生成
8. 出力層→ 10 種類の数字に分類

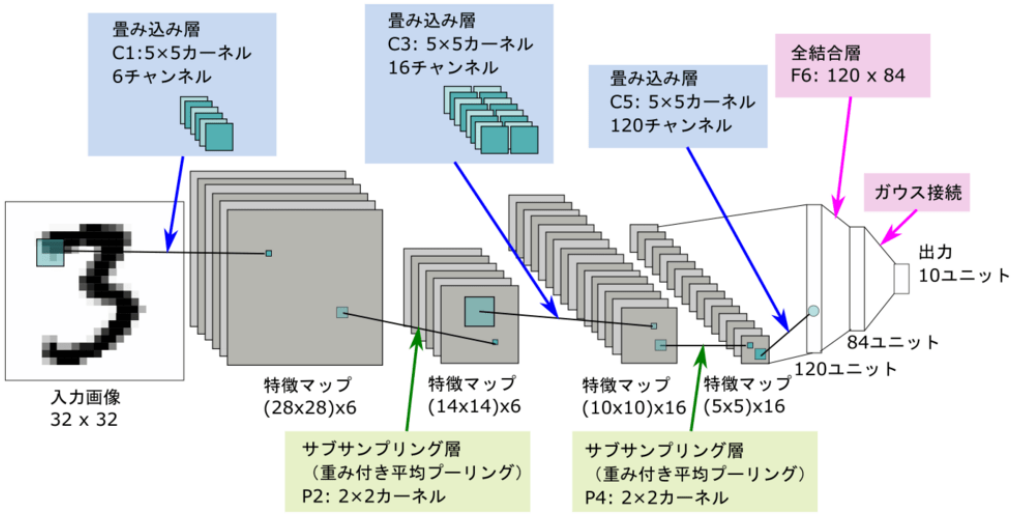


図 10: LeNet の概要 (https://cvml-expertguide.net/terms/dl/cnn/cnn-backbone/lenet/より引用)

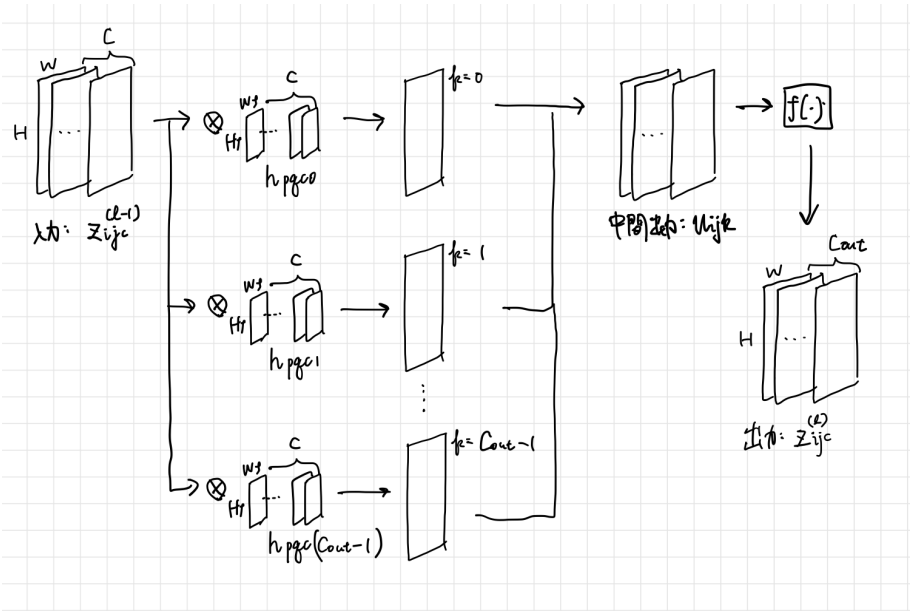


図 11: 畳み込み層の概念図. この畳み込み層は $W \times H \times C$ の入力に対し、C_{out} 個のフィルタ (縦横 $W_f \times H_f \times C$) を適用し、$W \times H \times C_{\text{out}}$ の出力を得る.

2.3 畳み込み (Convolution) 層

CNN では、LeNet 同様、畳み込み層、プーリング層、全結合層から構成される。CNN における畳み込み層では、フィルタ (カーネルとも呼ぶ) を用いて、画像という大きな行列の特徴を抽出する。フィルタは、画像の一部分に対して「重みのかけ合わせ = 畳み込み」を行い、画像の特徴を抽出するための行列である。

第 $l-1$ 層から $W \times H \times C$ の入力 $z_{ijc}^{(l-1)}$ を受け取り、C_{out} 種類のフィルタ $h_{pqck}(k = 0, \cdots, C_{\text{out}} - 1)$ を適用する場合を考える (図 11)。各フィルタのチャネル数は入力と同じ数 C であり、フィルタのサイズは $W_f \times H_f \times C$ と書ける。1 種類のフィルタに着目すると、入力チャンネル 1 つに対して、$W \times H \times 1$ の出力を得る。これをすべてのチャンネルで計算を行うと、$W \times H \times C$ の出力を得る。この出力をチャンネル方向で足し合わせて、1 種類のフィルタに対して、$W \times H \times 1$ の出力を得る。ここまでの計算は、以下で表すことができる。

$$u_{ijk}^{(l)} = \sum_{c=0}^{C-1} \sum_{p=0}^{W_f-1} \sum_{q=0}^{H_f-1} z_{i+p,j+q,c}^{(l-1)} h_{pqck} + b_k \tag{29}$$

$u_{ijk}^{(l)}$ は、k 番目のフィルタに対する出力である。ただし、b_k はバイアスである。この計算をすべてのフィルタに対して行うことで、$W \times H \times C_{\text{out}}$ の出力を得ることができる。この出力を活性化関数に通して、最終的な出力 $z_{ijk}^{(l)}$ を得る。

この積和演算を画像に対して行うことで、画像の特徴を抽出することができる。畳み込みでは、次元の繋がりを保ちながら、画像の特徴を抽出するために、フィルタをスライドさせて、画像全体に対して演算を行う。フィルタの数は、画像の特徴を抽出するための次元数であり、フィルタの数が多いほど、画像の特徴を多く抽出することができる。

畳み込み層における主要なハイパーパラメータには、以下のようなものがある。

- パディング
- ストライド
- チャンネル

パディングは、画像の次元を保つために画像の端にハイパーパラメータを追加することである。パディングを行うことで、フィルタのサイズ分小さくなる画像のサイズを元のサイズのまま保つことができる。

ストライドは、フィルタをスライドさせる幅を表すハイパーパラメータである。ストライドを大きくすることで、画像の次元を小さくすることができる。

チャンネルは、フィルタの数を表すハイパーパラメータである。チャンネルを増やすことで、画像の特徴を多く抽出することができる。

確認テスト

Q: サイズ 6×6 の入力画像を、サイズ 2×2 のフィルタで畳み込んだ時の出力画像のサイズを答えよ。なおストライドとパディングは 1 とする。

A: 出力画像は 5×5 となる。なぜなら、ストライド 1 の時、フィルタをスライドさせる間隔は 1 つずつで、パディングが 1 の時、入力画像の周囲に 1 マスずつ余白が生じるためである。

なお、公式は以下の通りとなる。

$$\frac{\text{入力画像のサイズ} + 2 \times \text{パディング} - \text{フィルタサイズ}}{\text{ストライド}} + 1 \tag{30}$$

2.4 プーリング (pooling) 層

一般的にプーリング層では 2 つの役割がある。1 つは、入力の各位置でその局所領域内の値を要約することであり、もうひとつは入力の空間解像度を下げるダウンサンプリングの実行である。要約の仕方は、畳み込んだ行列の中で最大の値を取り出したり、平均値を取り出すなどがある。この時、前者の方法を Max プーリング、後者の方法を Average プーリングと呼ぶ。

2.5 全結合層

全結合層は、畳み込み層やプーリング層で抽出した特徴を元に、最終的な出力を行う層である。全結合層では、畳み込み層やプーリング層で抽出した特徴を 1 次元のベクトルに変換し、出力層に渡す。数式では単にアフィン変換を行うだけであるが、この層がないと、畳み込み層やプーリング層で抽出した特徴を元に、最終的な出力を行うことができない。

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{31}$$

全結合層はしばしば線形層、Linear クラスまたは単に Linear と記述されることが多い。

1 次元への変換は、Flatten, Global MaxPooging, Global AveragePooling などの手法がある。Flatten は、抽出した特徴をすべて 1 次元のベクトルに変換する手法である Global MaxPooling は、抽出した特徴の中から 1 チャンネルの中での最大の値を取り出す手法である。Global AveragePooling は、抽出した特徴の中から 1 チャンネルの中での平均値を取り出す手法である。

2.6 AlexNet

AlexNet は、深層学習による画像認識ブームの火付け役となったモデルである。224×224 の画像を入力とし、55×55 の畳み込み層、27×27 のプーリング層、13×13 の畳み込み層、6×6 のプーリング層、全結合層、出力層から構成される。AlexNet では、以下の工夫を行ったことで、誤差が大幅に削減され、高い性能を発揮した。

1. ReLU 関数の導入
2. ドロップアウトの導入
3. データ拡張
4. バッチ正規化

それまでの CNN では、シグモイド関数やハイパボリックタンジェント関数を活性化関数として用いていたが、これらの関数を用いた場合、層を増やした時に微分値が小さくなるため、勾配が爆発したり消失する問題があった。ReLU 関数は、そのような問題を解決するために導入された関数であり、勾配が消失しないため、層を深くすることができた。

2.7 GoogLeNet

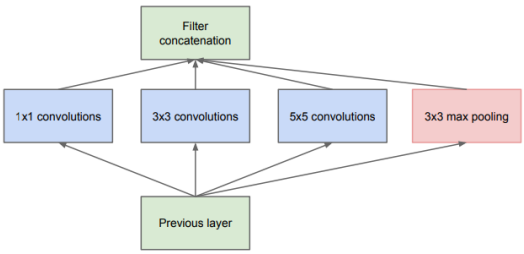
GoogLeNet は、2014 年に発表されたモデルであり、当時としてはとても層が多く、深いネットワークを有するモデルであった。その深いネットワークを実現するために、様々な問題を回避しつつ、層数を増やした点

が現代においても評価されている。以下に、その工夫点を示す。

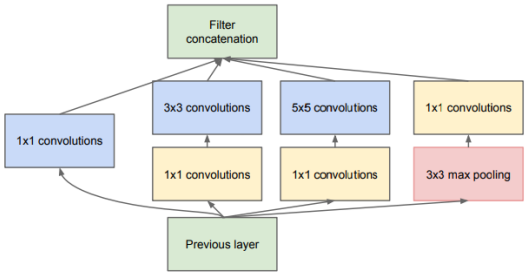
1. Inception モジュールの導入
2. Auxilary Loss の導入
3. Alobal Average Pooling の導入

2.7.1 Inception モジュール

Inception モジュールでは、異なるフィルタサイズを持つ畳み込み層を並列に組み合わせることで、層を深くすることなくフィルターの種類を増やすことができる。フィルターの種類を増やすことで、異なる空間分解能で特徴抽出を行うので、より性能の高いモデルを構築することができる。1×1 の畳み込み層は、Pointwise Convolution と呼ばれ、次元削減を行うために用いられる。入力のチャンネル数よりも少ない種類のフィルターを用いることで、チャンネル方向でまとめられた特徴をフィルターの数分出力することができる。



(a) シンプルな Inception モジュール (出典: https://arxiv.org/pdf/1409.4842)



(b) 次元削減を伴う Inception モジュール (出典: https://arxiv.org/pdf/1409.4842)

図 12: Inception モジュールの概略図

2.7.2 Auxilary Loss

Auxilary Loss は、GoogLeNet において、途中の層で出力を行い、その出力に対して損失関数を適用する手法である。この手法は、勾配消失問題を回避するために導入された手法であり、途中の層で出力を行うことで、途中の層での勾配が消失することを防ぐことができる。

2.7.3 Global Average Pooling

Global Average Pooling は、畳み込み層やプーリング層で抽出した特徴を、チャンネル方向に平均を取る手法である。この手法は、畳み込み層やプーリング層で抽出した特徴を 1 次元のベクトルに変換することができるため、全結合層に入力することができる。

■参考文献

1. 岡谷貴之/深層学習 改訂第 2 版 [機械学習プロフェッショナルシリーズ]/ 講談社サイエンティフィク/ 2022-01-17
2. バッチ正規化 (Batch Normalization) とその発展形 https://cvml-expertguide.net/terms/dl/layers/batch-normalization-layer/

3. 全結合層 (fully-connected layer) [線形層] https://cvml-expertguide.net/terms/dl/layers/fully-connected-layer/
4. LeNet: 最初の CNN 構造 https://cvml-expertguide.net/terms/dl/cnn/cnn-backbone/lenet/
5. AlexNet: 大規模な画像物体認識むけ CNN の元祖 https://cvml-expertguide.net/terms/dl/cnn/cnn-backbone/alexnet/#3_AlexNet_%E3%81%AE%E7%89%B9%E5%BE%B4%E3%83%BB%E5%B7%A5%E5%A4%AB
6. 【要点を整理】GoogleNet をわかりやすく解説 https://dx-consultant-fast-evolving.com/googlenet/
7. Going deeper with convolutions https://arxiv.org/pdf/1409.4842