

1 Section1_勾配消失問題

1. 1 全体像

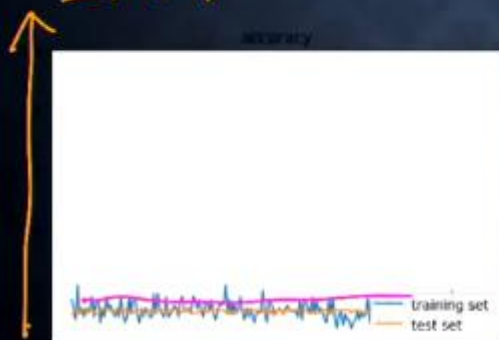
勾配消失問題の復習

勾配消失問題

誤差逆伝播法が下位層に進んでいくに連れて、勾配がどんどん緩やかになっていく。そのため、勾配降下法による、更新では下位層のパラメータはほとんど変わらず、訓練は最適値に収束しなくなる。

正解率

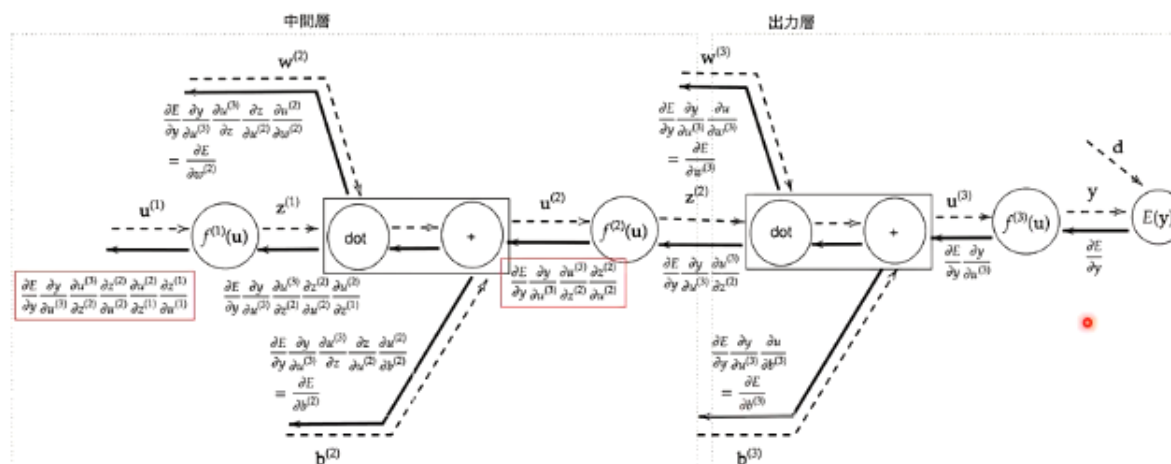
正解率



何回学習したか

何回学習したか

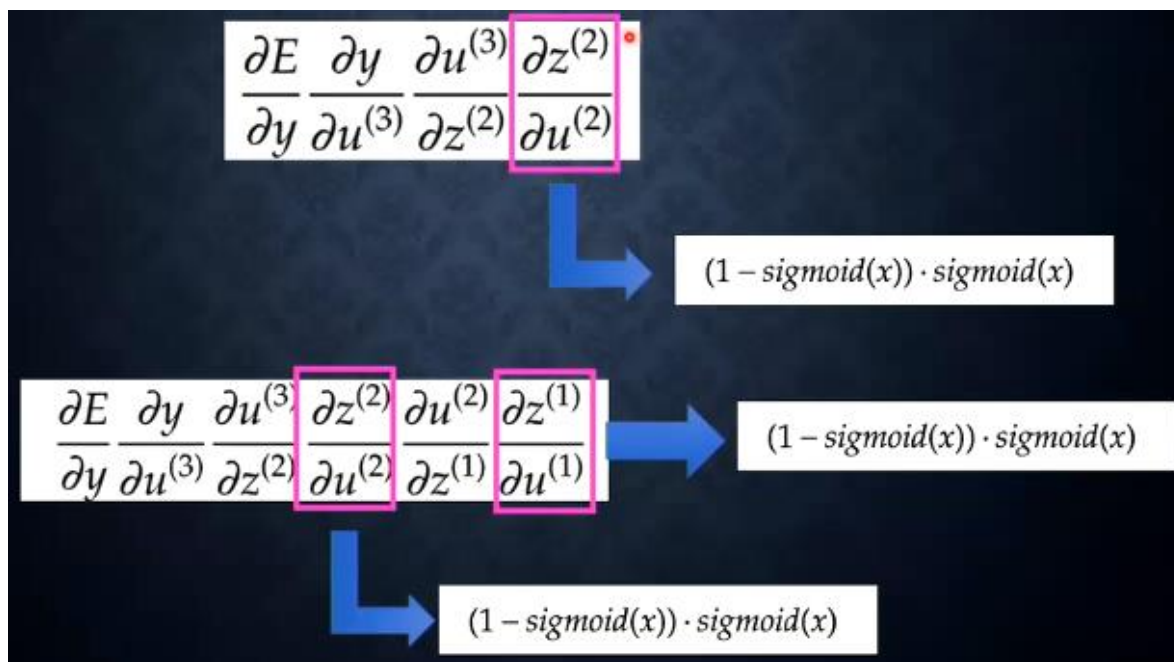
1) 勾配消失問題のビジョン



微分値は0～1の値を取るものが増えてくる。掛けるとどんどん小さくなる。

(2) 活性化関数：シグモイド関数

シグモイド関数の微分値は最大で0.25となる。



(3) 勾配消失の解決法

- ①活性化関数の選択
- ②重みの初期値設定
- ③バッチ正規化

1. 2 活性化関数

(1) ReLU 関数

微分結果： $0 < \text{時}$ は 0、 ≥ 0 時は 1

(2) 結果

①勾配消失問題が解消される

②微分値が 0 となった時の重みは使用されない（スパース化）

(3) 重みの初期値設定 Xavier（ザビエル）

多くの場合は乱数を使用する。

Xavier の初期値を設定する際の活性化関数

①ReLU 関数

②シグモイド（ロジスティック）関数

③双曲線正接関数

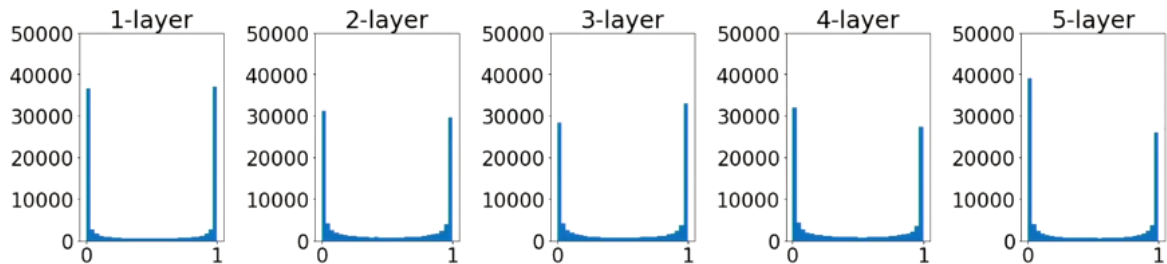
標準正規分布に基づいて初期化する（平均が 0、分散が 1）

```
network[W1] = np.random.randn(input_layer_size, hidden_layer_size) / np.sqrt(input_layer_size)
network[W2] = np.random.randn(hidden_layer_size, output_layer_size) / np.sqrt(hidden_layer_size)
```

●重みの要素を、前の層のノード数の平方根で除算した値

1. 3 初期値の設定方法

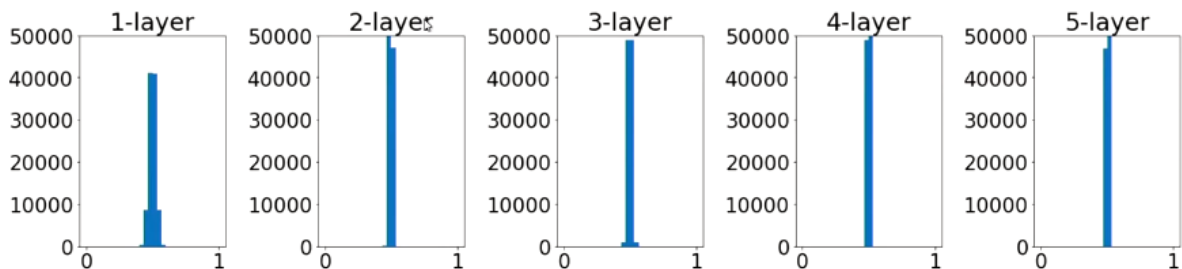
(1) 標準正規分布で重み付け



1 か 0 を出力

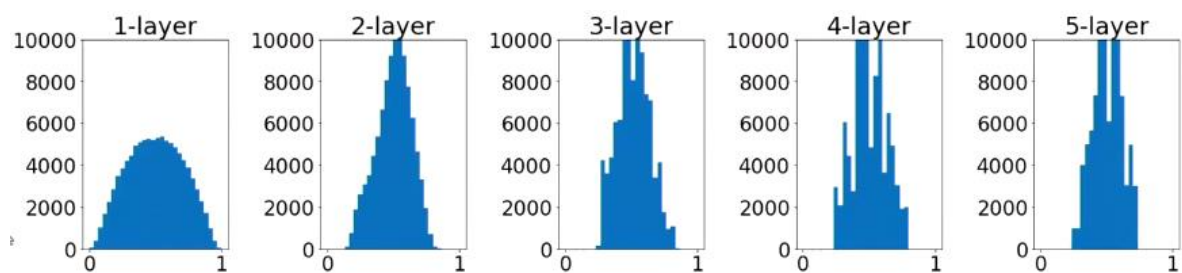
逆伝播に向かない

(2) 適当に小さい値で割る



0.5 による

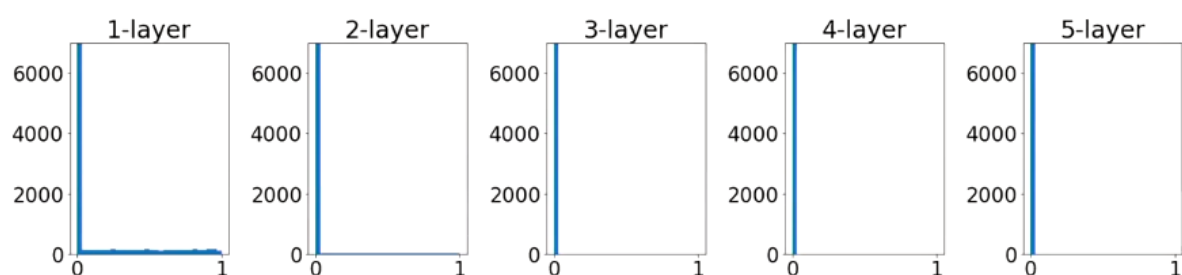
(3) Xavier の初期化



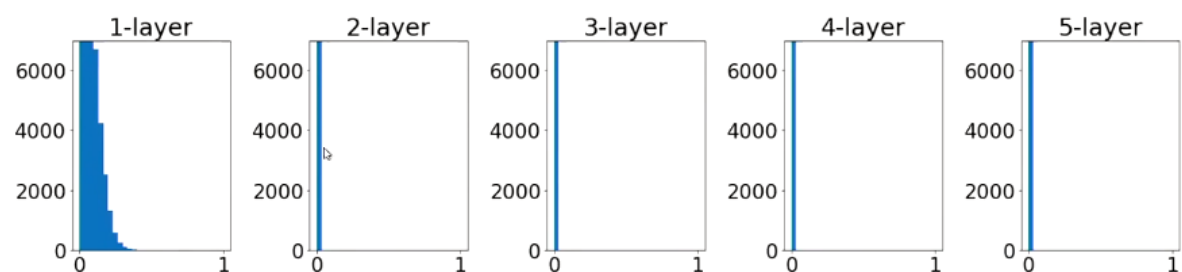
S 字カーブの関数によく働く

(4) 初期設定値を He 初期化 (ReLU 関数)

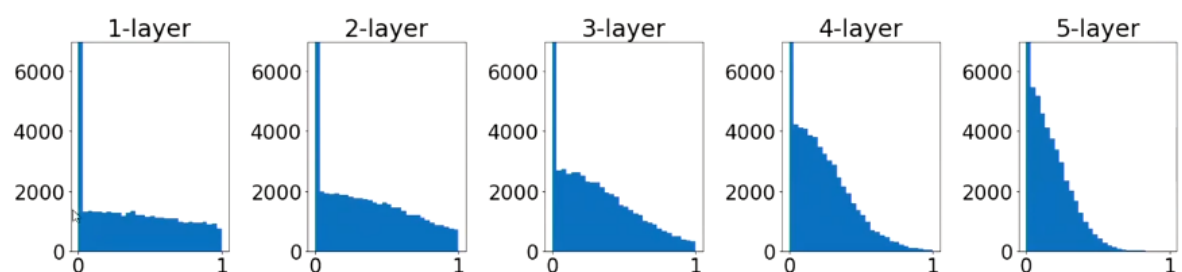
①標準偏差



②標準偏差を値を小さくした



③He 初期化



(5) 重みの初期値を 0 にする

☆重みを 0 で初期化すると正しい学習が行えない
→ すべての重みの値が均一に更新されるため、
多数の重みを持つ意味がなくなる。

1. 4 バッチ正規化

(1) バッチ正規化とは

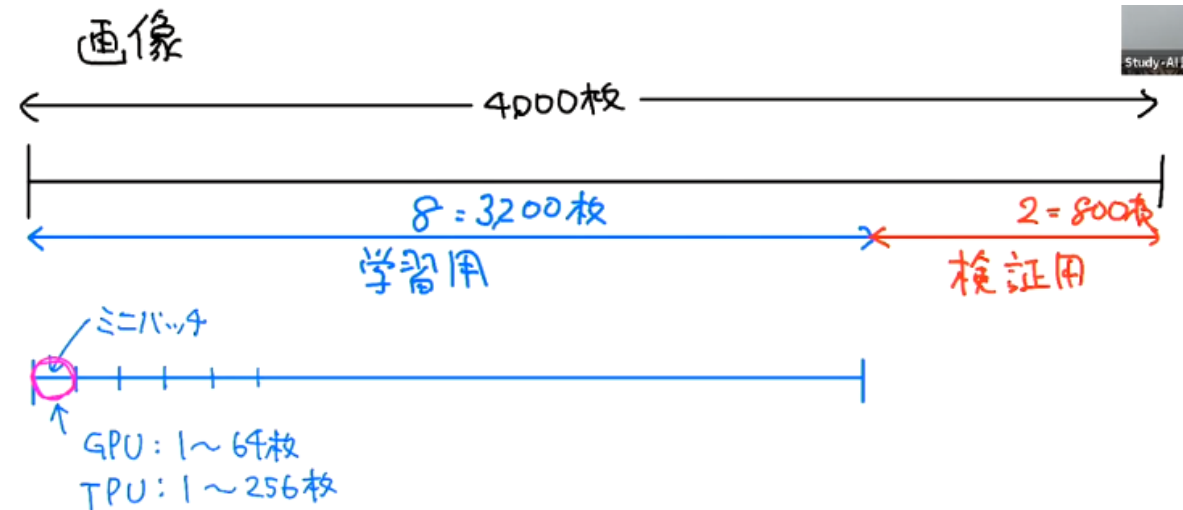
ミニバッチ単位で、入力値のデータの偏りを抑制する手法

(2) バッチ正規化の使いどころとは

活性化関数に値を渡す前後に、バッチ正規化の処理を孕んだ層を加える。

バッチ正規化層への入力値は

$$u = wz + b \text{ 又は } z$$



機材 { CPU ... 汎用
GPU ... ゲーム用 → AI
TPU ... AI専用

- ・中間層の重みの更新がうまくいく
- ・過学習が起きにくい
- ・ミニバッチである程度正規化する。極端な分布がなくなる。

(3) 数学的記述

ミニバッチの平均

$$1. \mu_t = \frac{1}{N_t} \sum_{i=1}^{N_t} x_{ni}$$

ミニバッチの分布

$$2. \sigma_t^2 = \frac{1}{N_t} \sum_{i=1}^{N_t} (x_{ni} - \mu_t)^2$$

ミニバッチの正規化

$$3. \hat{x}_{ni} = \frac{x_{ni} - \mu_t}{\sqrt{\sigma_t^2 + \theta}}$$

変倍移動

$$4. y_{ni} = \gamma \hat{x}_{ni} + \beta$$

(4) 処理及び記号の説明

μ_t : ミニバッチ t 全体の平均
 σ_t^2 : ミニバッチ t 全体の標準偏差
 N_t : ミニバッチのインデックス
 \hat{x}_{ni} : 0に値を近づける計算(0を中心とするセンタリング)と正規化を施した値
 γ : スケーリングパラメータ
 β : シフトパラメータ
 y_{ni} : ミニバッチのインデックス値とスケーリングの積にシフトを加算した値(バッチ正規化オペレーションの出力)

記述

ミニバッチ 8枚
↓
x

ミニバッチの平均 →

ミニバッチの分散 →

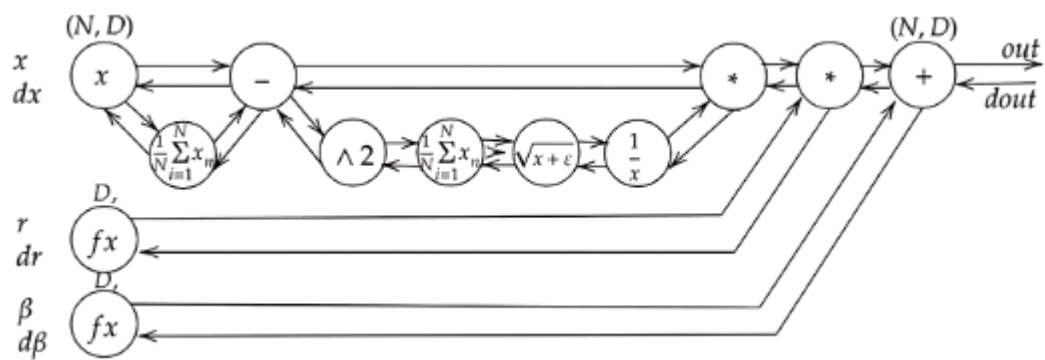
ミニバッチの正規化 →

変換・移動 →

1. $\mu_t = \frac{1}{N_t} \sum_{i=1}^{N_t} x_{ni}$
2. $\sigma_t^2 = \frac{1}{N_t} \sum_{i=1}^{N_t} (x_{ni} - \mu_t)^2$
3. $\hat{x}_{ni} = \frac{x_{ni} - \mu_t}{\sqrt{\sigma_t^2 + \theta}}$
4. $y_{ni} = \gamma x_{ni} + \beta$

統計的
正規化

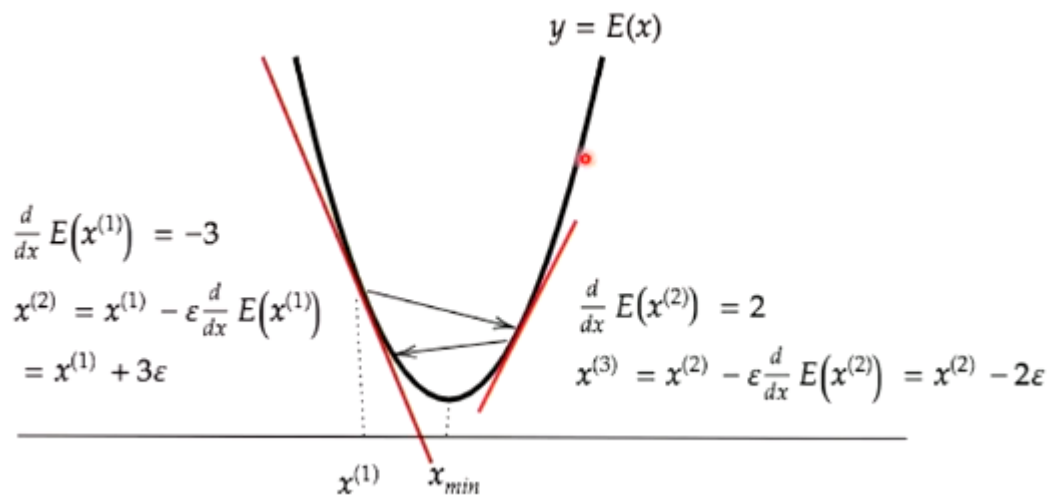
記号の説明



2 Section2_学習率最適化手法

2. 1 全体像

誤差関数の最小値を検索する。



学習率を可変するとどのようなことが起こるのか

(1) 学習率が大きい場合

①最適値にいつまでもたどり着かず、発散してしまう。

(2) 学習率が小さい場合

②発散することはないが、小さすぎると収束するまでに時間がかかる。

③大域局所最適値に収束しづらくなる。

(3) 学習率の決め方とは

初期の学習率設定方法の指針

①初期の学習率を大きく設定し、徐々に学習率を小さくしていく

②パラメータごとに学習率を可変させる。

→学習率最適化手法を利用して学習率を最適化

2. 2 モメンタム

モメンタム

勾配降下法

$$V_t = \mu V_{t-1} - \epsilon \nabla E$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \nabla E$$

```
self.v[key] = self.momentum * self.v[key] - self.learning_rate * grad[key]
```

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + V_t$$

```
params[key] += self.v[key]
```

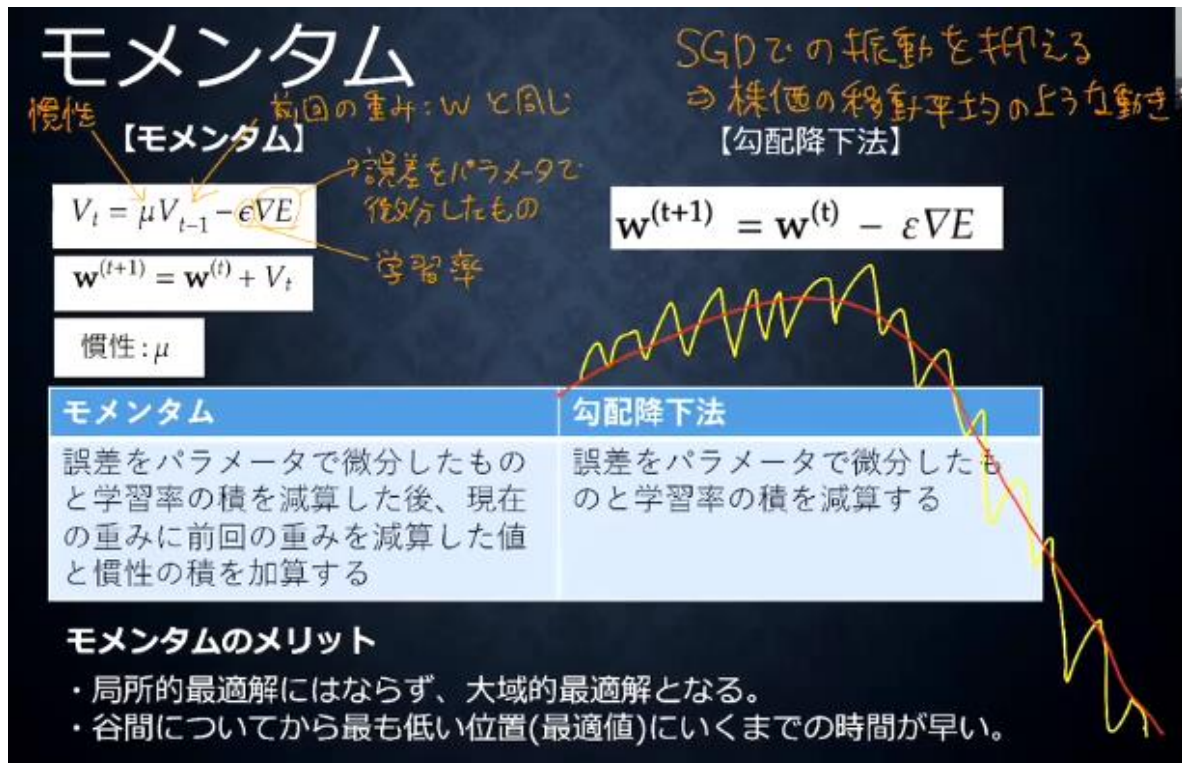
慣性： μ

誤差をパラメータで微分したものと学習率の積を減算した後、現在の重みに前回の重みを減算した値と慣性の積を加算する

誤差をパラメータで微分したものと学習率の積を減算する

(1) モメンタムのメリット

- ① 局所的最適解にならず、大域的最適解になる。
- ② 谷間についてから最も低い位置（最適値）に行くまでの時間が早い。



2. 3 Adagrad

$$h_0 = \theta$$

```
self.h[key] = np.zeros_like(val)
```

$$h_t = h_{t-1} + (\nabla E)^2$$

```
self.h[key] += grad[key] * grad[key]
```

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$$

```
params[key] -= self.learning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e-7)
```

誤差をパラメータで微分したものと
再定義した学習率の積を減算する

(1) Adagrad のメリット

①勾配の緩やかな斜面に対して、最適値に近づける

(2) 課題

学習率が徐々に小さくなるので、鞍点問題を引き起こすことが有った。

数式とコード

$h_0 = \theta$ ← なにかしらの値で h を初期化

```
self.h[key] = np.zeros_like(val)
```

$h_t = h_{t-1} + (\nabla E)^2$ ← 計算した勾配の2乗を保持

```
self.h[key] += grad[key] * grad[key]
```

$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$ ← 現在の重みを 適応させた学習率で更新

```
params[key] -= self.learning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e-7)
```

2. 4 RMSProp

$$h_t = \alpha h_{t-1} + (1 - \alpha) (\nabla E)^2$$

$\alpha : 0 \sim 1$

```
self.h[key] *= self.decay_rate  
self.h[key] += (1 - self.decay_rate) * grad[key] * grad[key]
```

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t + \theta}} \nabla E$$

```
params[key] -= self.learning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e-7)
```

誤差をパラメータで微分したものと
再定義した学習率の積を減算する

(1) RMSProp のメリット

- ①局所的最適解にならず、大域的最適解になる。
- ②ハイパーパラメータの調整が少ない。
- ③鞍点問題を解消

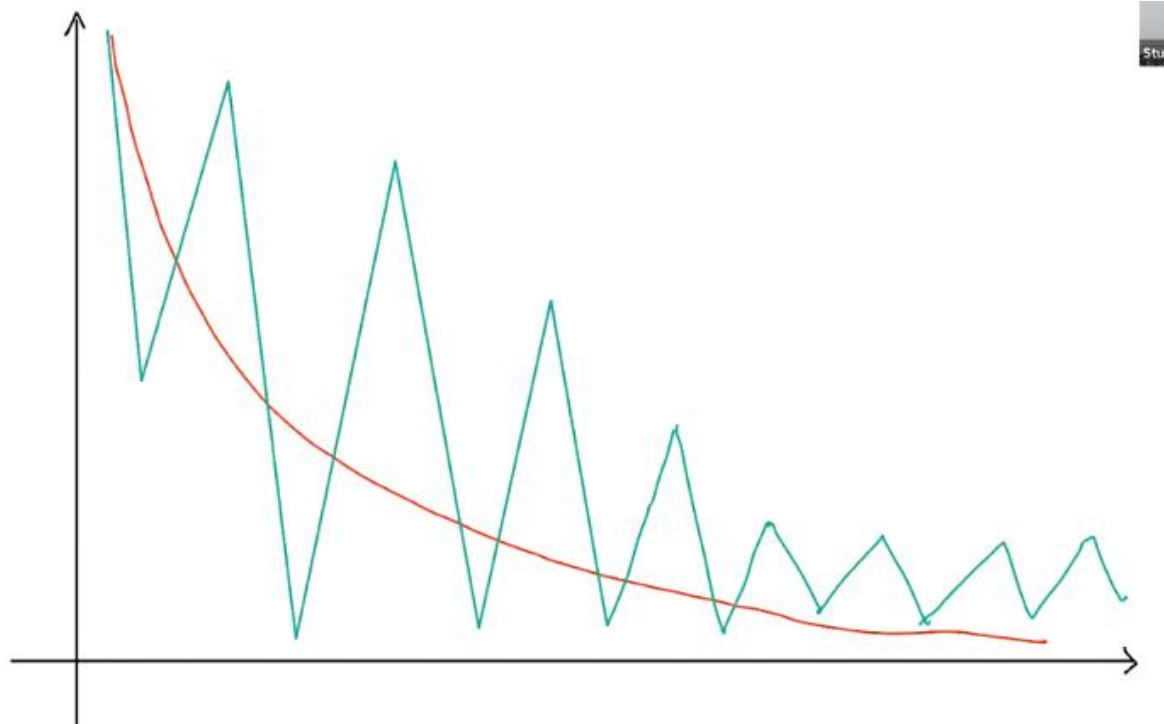
2. 5 Adam

1) Adam とは

- ①. モメンタムの、過去の勾配の指数関数的減衰平均
 - ②. RMSProp の、過去の勾配の2乗の指数関数的減衰平均
- 上記をそれぞれ孕んだ最適化アルゴリズムである。

2) Adam のメリットとは

- ①. モメンタムおよび RMSProp のメリットを孕んだアルゴリズムである。

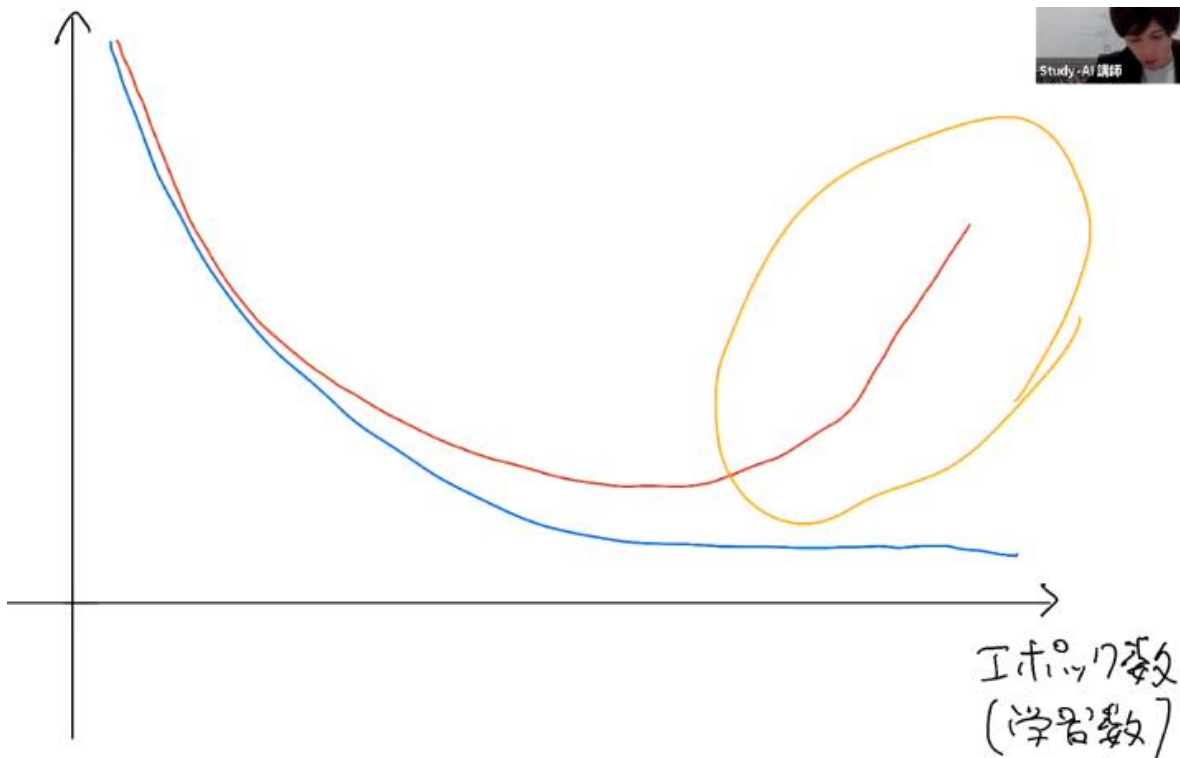
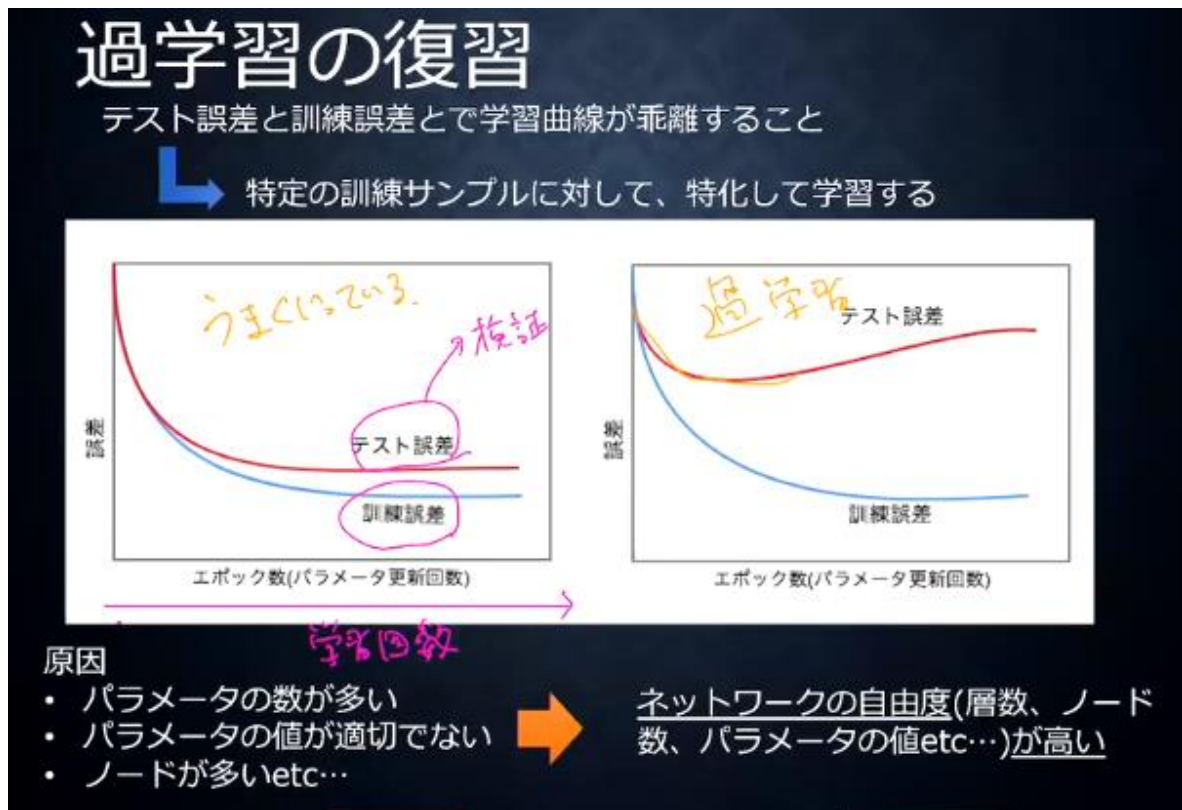


3 Section3_過学習

3. 1 全体像

(1) 過学習の原因

- ①パラメータの数が多い
- ②入力値が少ない。パラメータが多い。ノード数が多い。



ニューラルネットワークの自由度が高過ぎる

3. 2 正規化手法 1

(1) 正則化とは

①ネットワークの自由度（層数、ノード数、パラメータ数 etc...）を制約すること。

→正則化手法を利用して過学習を抑制する。

(2) 正則化手法について

①L1 正則化、L2 正則化

②ドロップアウト

(3) Weight decay（荷重減衰）

①過学習の原因

・重みが大きい値を取ることで、過学習が発生することがある。

→学習させていくと、重みにバラつきが発生する。

重みが大きい値は、学習において重要な値であり、重みが大きいと過学習が起こる。

②過学習の解決策

・誤差に対して、正則化項を加算することで、重みを抑制する。

→過学習が起こりそうな重みの大きさ以下で重みをコントロールし、かつ重みの大きさにバラつきを出す必要がある。

(4) L1 正則化、L2 正則化

$$E_n(\mathbf{w}) + \frac{1}{p} \lambda \|\mathbf{x}\|_p$$

: 誤差関数に、p ノルムを加える（距離）

```
np.sum(np.abs(network.params['W' + str(idx)]))
```

$$\|\mathbf{x}\|_p = \left(|x_1|^p + \dots + |x_n|^p \right)^{\frac{1}{p}}$$

: p ノルムの計算

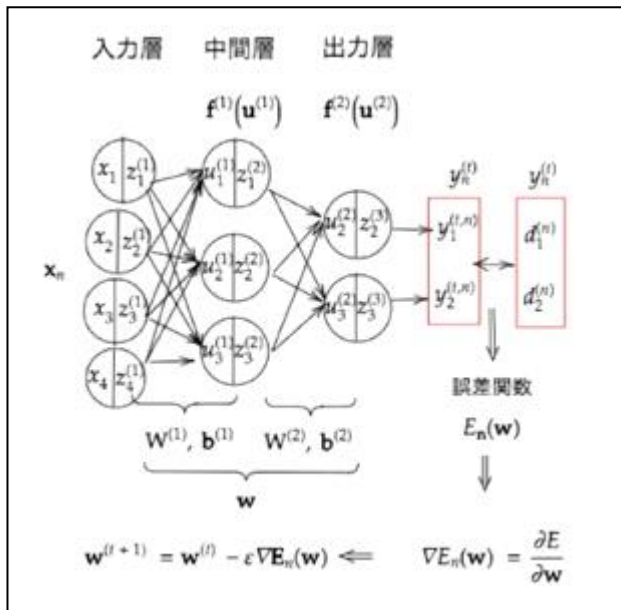
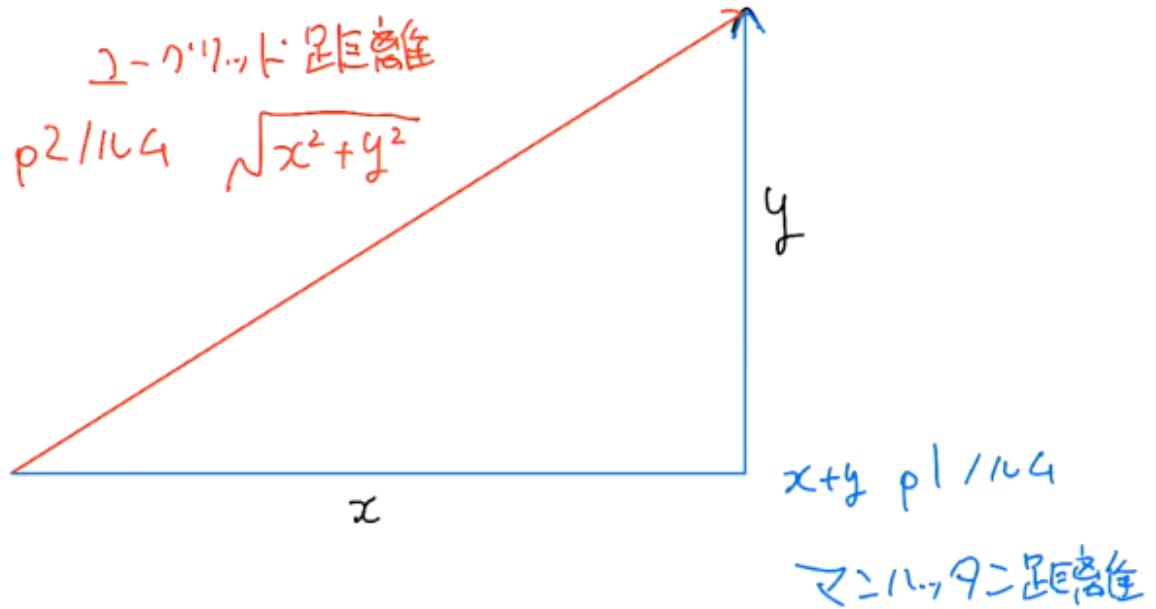
```
weight_decay += weight_decay_lambda
* np.sum(np.abs(network.params['W' + str(idx)]))

loss = network.loss(x_batch, d_batch) + weight_decay
```

p = 1 の場合、L1 正則化と呼ぶ ラッソ回帰

p = 2 の場合、L2 正則化と呼ぶ リッジ回帰

3. 3 正規化手法 2



$$\| \mathbf{w}^{(1)} \|_p = \left(|w_1^{(1)}|^p + \dots + |w_n^{(1)}|^p \right)^{\frac{1}{p}}$$

$$\| \mathbf{w}^{(2)} \|_p = \left(|w_1^{(2)}|^p + \dots + |w_n^{(2)}|^p \right)^{\frac{1}{p}}$$

$$\| \mathbf{x} \|_p = \| \mathbf{w}^{(1)} \|_p + \| \mathbf{w}^{(2)} \|_p$$

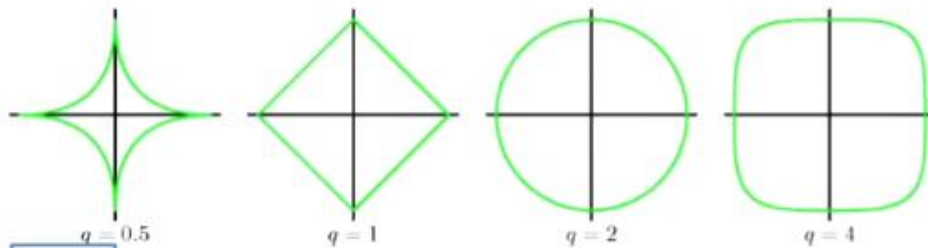
$$E_n(\mathbf{w}) + \frac{1}{p} \lambda \| \mathbf{x} \|_p$$

いろいろな正規化項のqとグラフ

- 正規化項のqを変化させて
M=2のとき、グラフにすると以下になる $\sum_{j=1}^M |w_j|^q$



Rで描いた三次元グラフ



補足

PRML

等高線の形(三次元グラフを横で切った形)

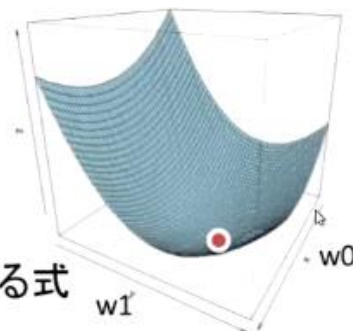
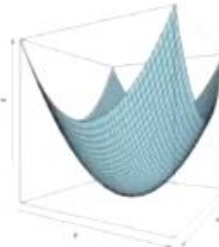
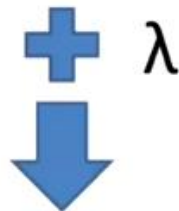
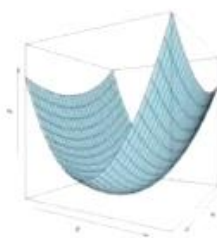
44

q=2のときの最小化



誤差関数

正則化項



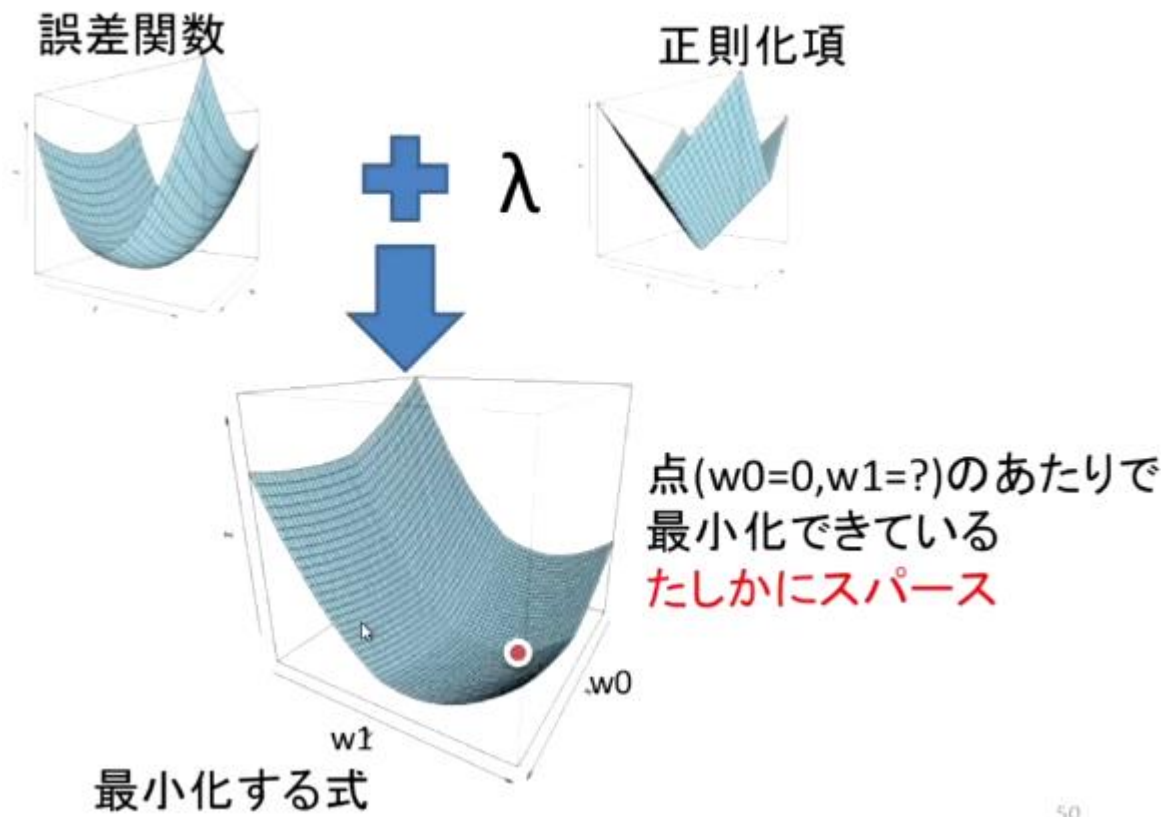
点($w_0=?$, $w_1=?$)のあたりで
最適解になっている
どうみてもスパースでない

最小化する式

補足

49

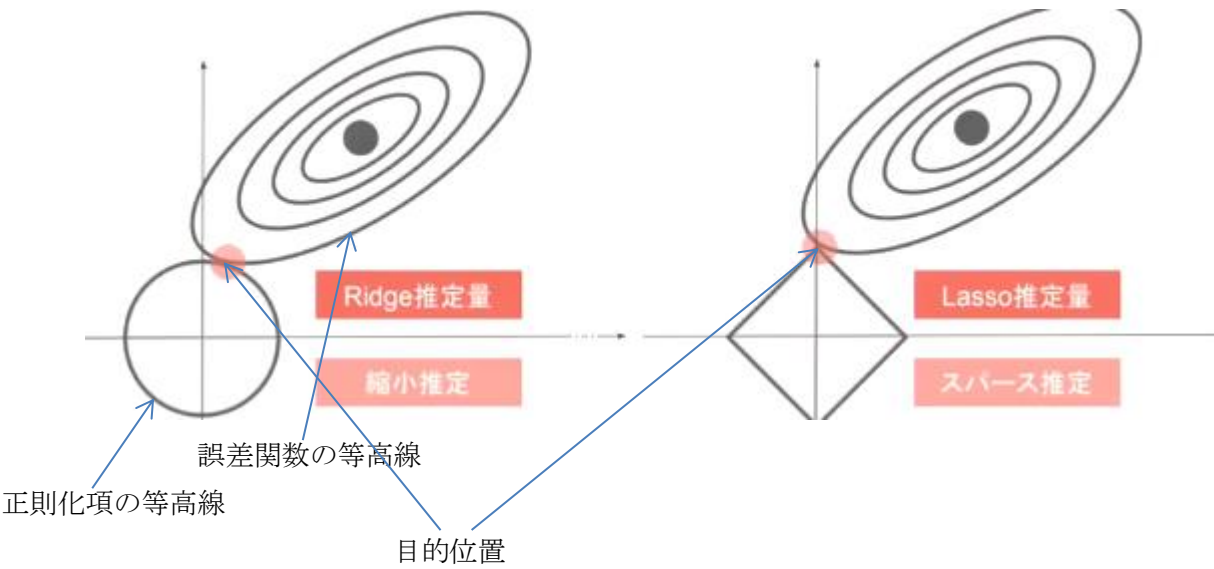
lassoのときの最小化



50

角が出てくる。過度になると重みが0となる。
ReLU と同じようなことが起きる。

3. 5 正規化手法 4

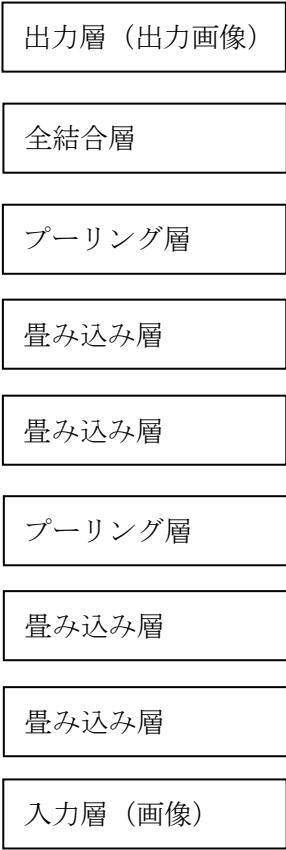


4 Section4_畳み込みニューラルネットワークの概念

4. 1 構造 1

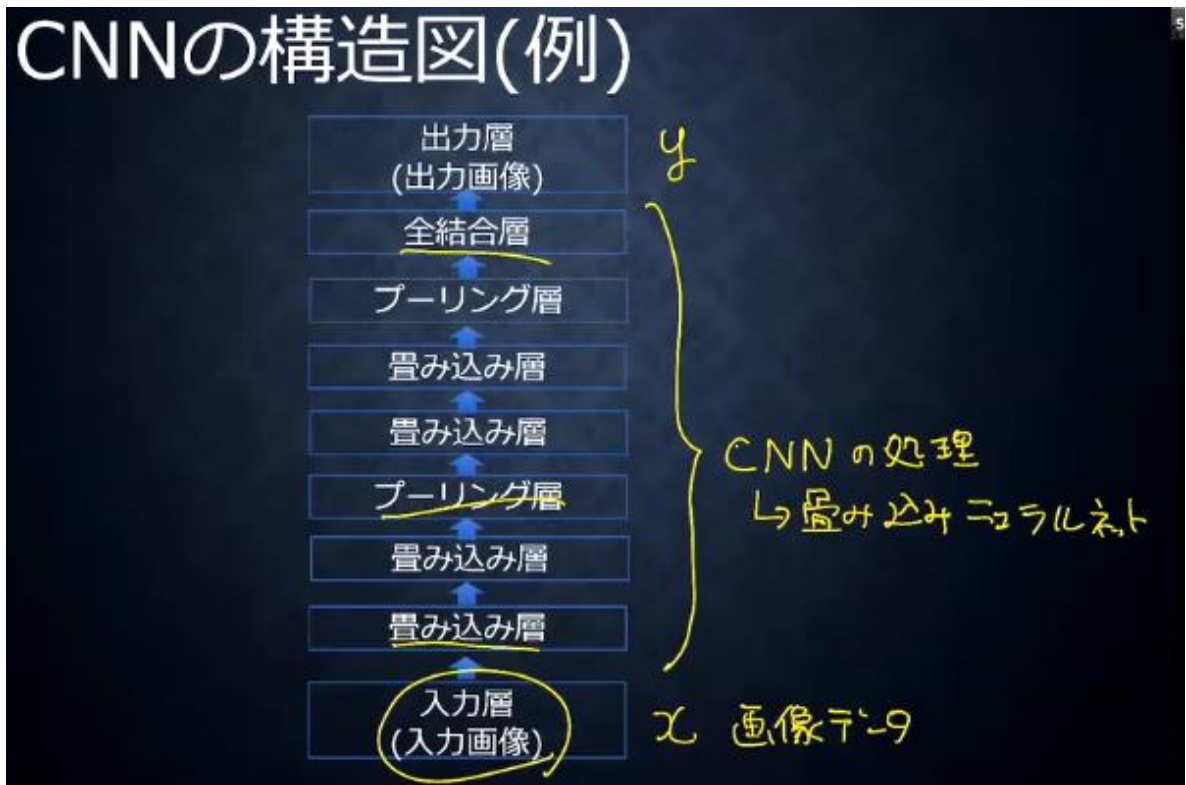
(1) CNN の構造図 (例)

CNN は次元間でつながりのあるデータを扱える

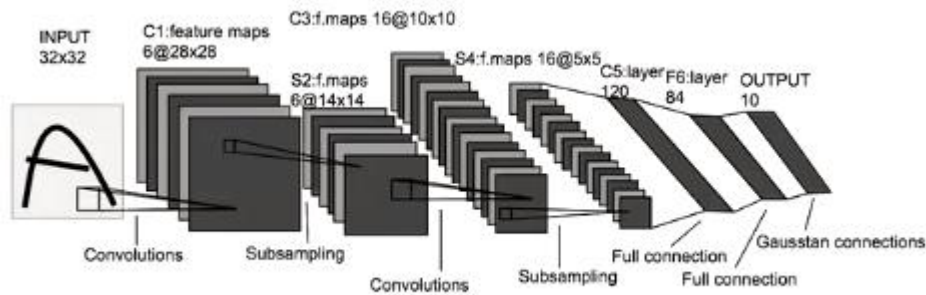


	1 次元	2 次元	3 次元
単一チャネル	音声 (時刻、強度)	フーリエ変換した音声 (時刻、周波数、強度)	CT スキャン画像
複数チャネル	アニメのスケルトン (時刻、腕の価、膝の価、...)	カラー画像 (x、y、(R,G,B))	動画 (時刻、x、y、(R,G,B))

4. 2 構造 2

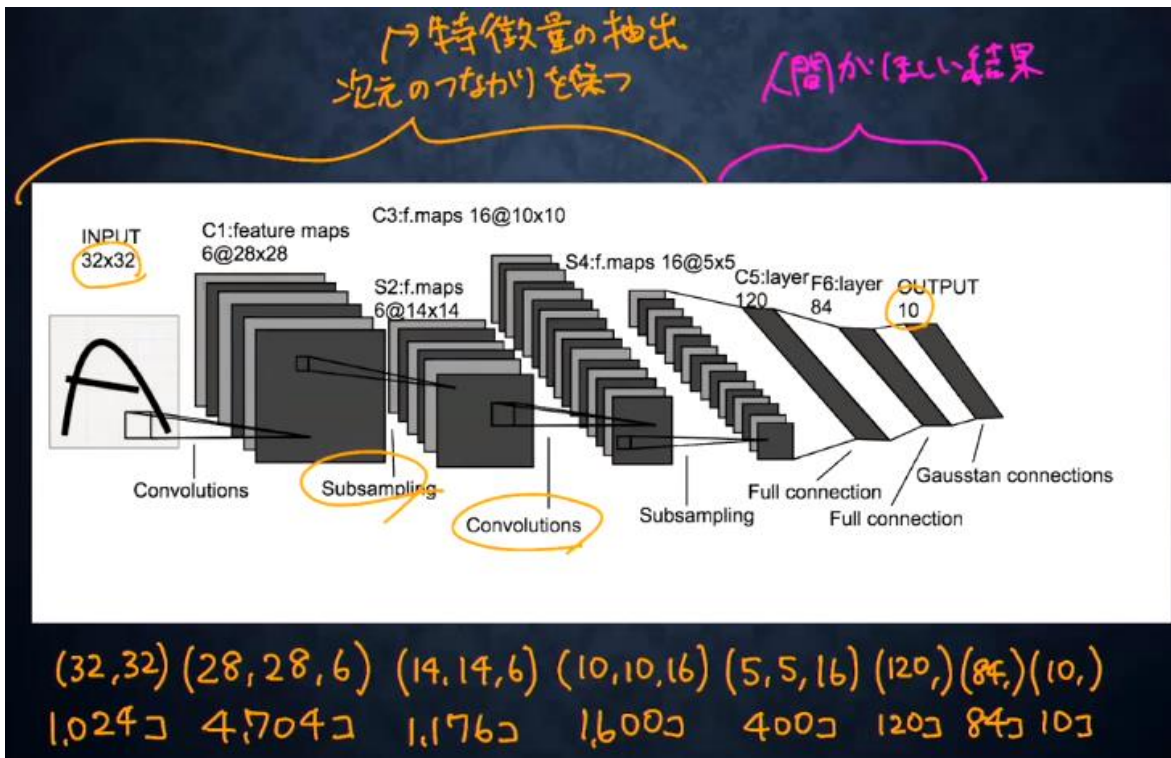


(1) LeNet の構造図



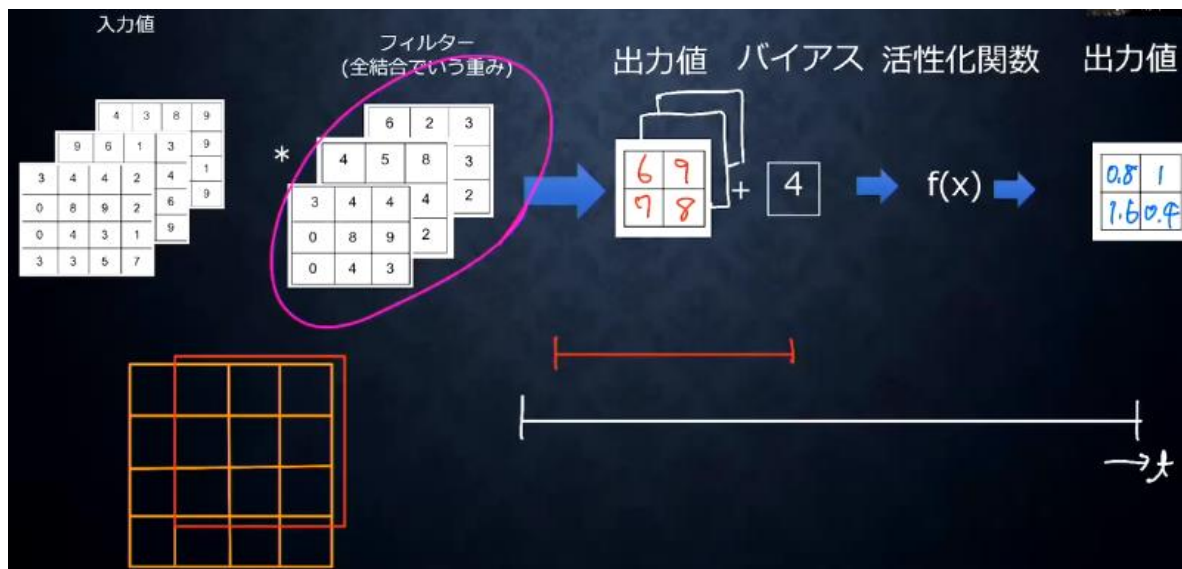
(32,32,3) (28,28,6) (14,14,6) (10,10,16) (5,5,16) (120,) (84,) (10,)

1024 4704 1176 1600 400



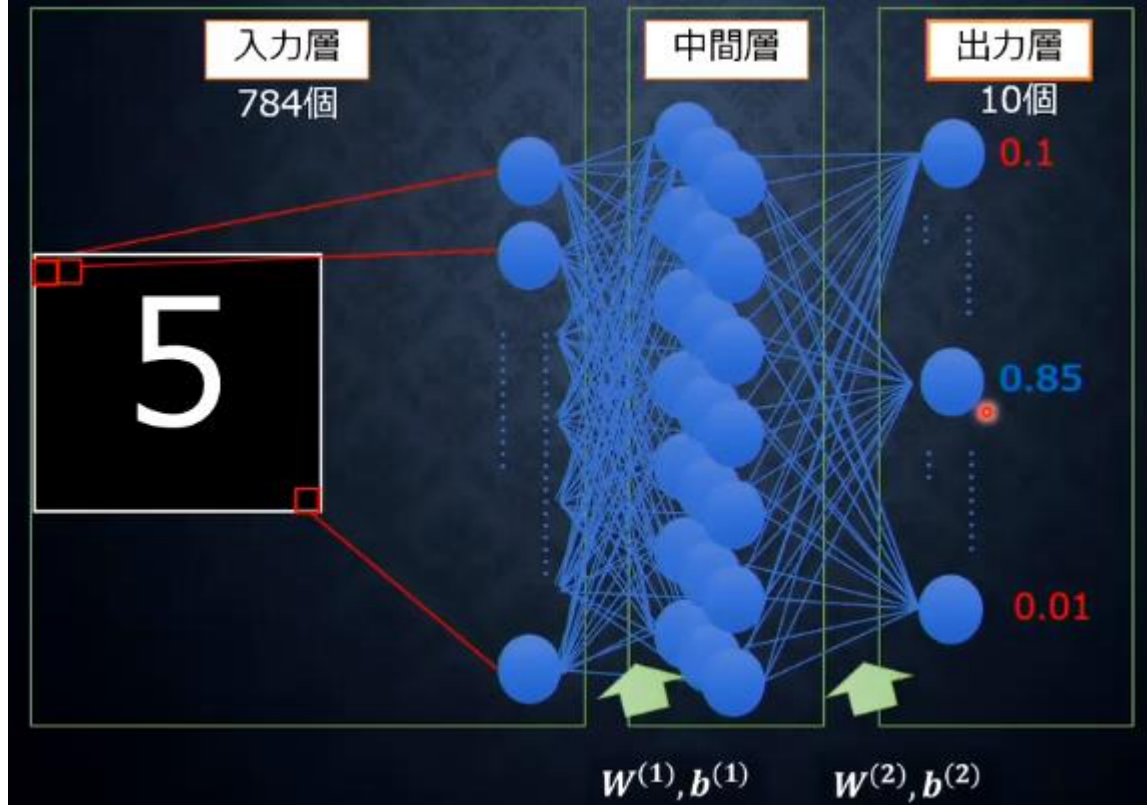
4. 3 全体像

(1) 畳み込み層



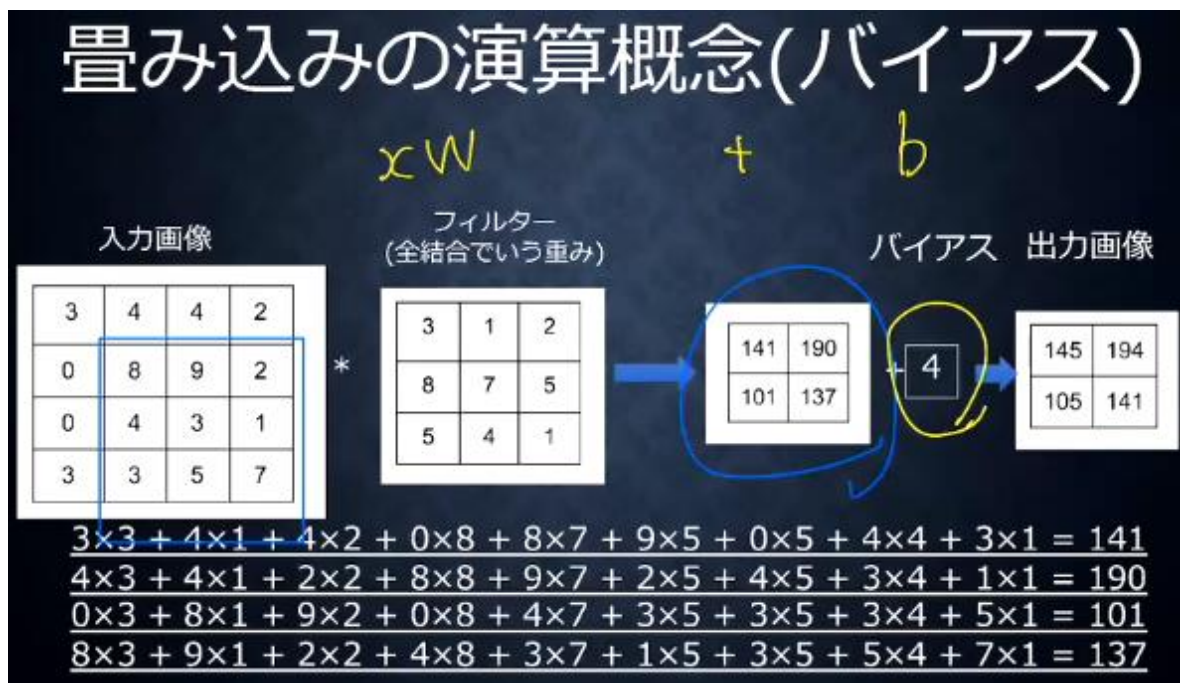
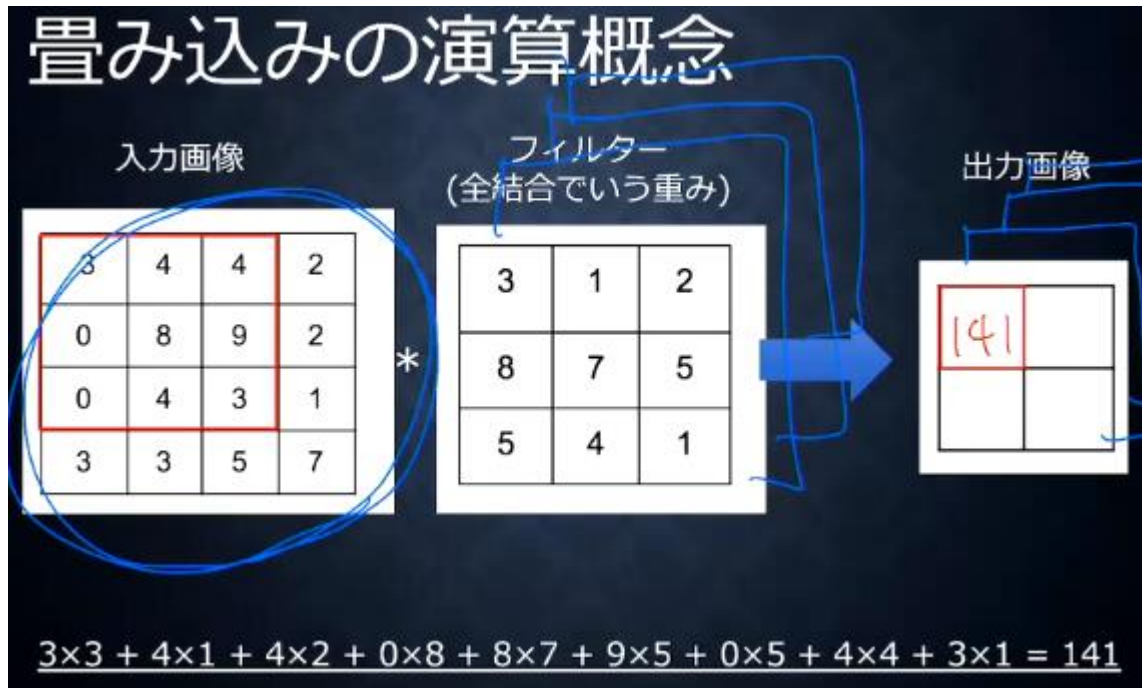
次元間の繋がりが保たれている。

全結合での学習



4. 4 畳み込み層（バイアス）

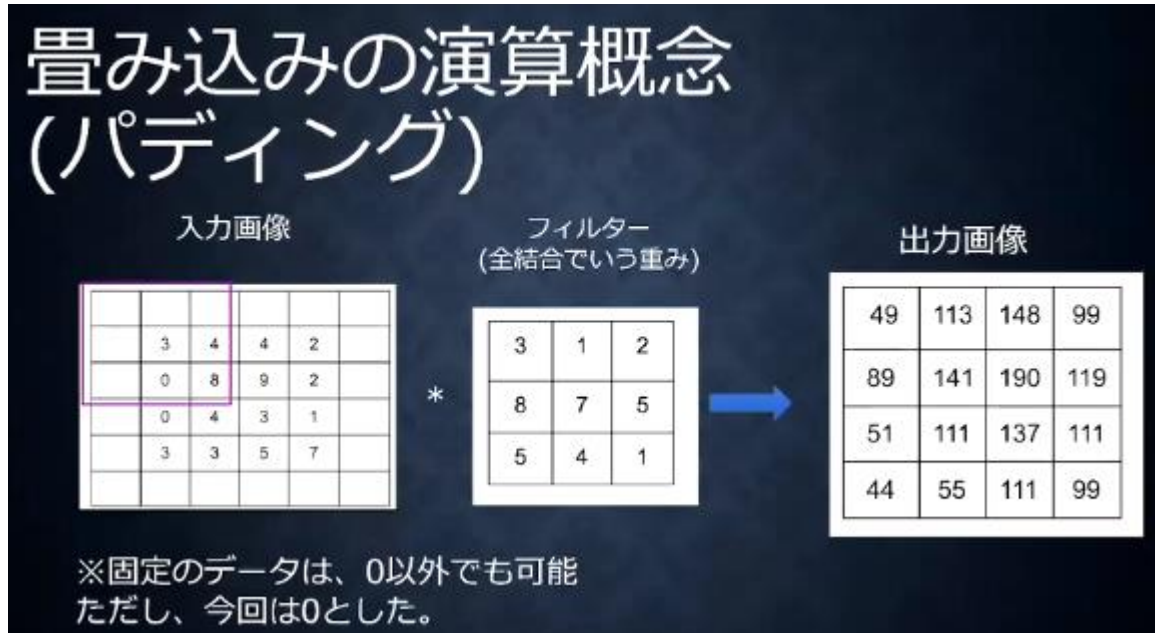
- (1) 畳み込み層では、画像の場合、縦、横、チャンネルの3次元のデータをそのまま学習し、次に伝えることができる。
- (2) 結論：3次元の空間情報も学習できるような層が畳み込み層である。



(3) パディング

- 4×4 の画像を 3×3 のフィルターを通すと、 2×2 となる。
このため、パディングを行い、出力画像の画素数を減らないようにする。

4. 5 畳み込み層（パディング・ストライド）

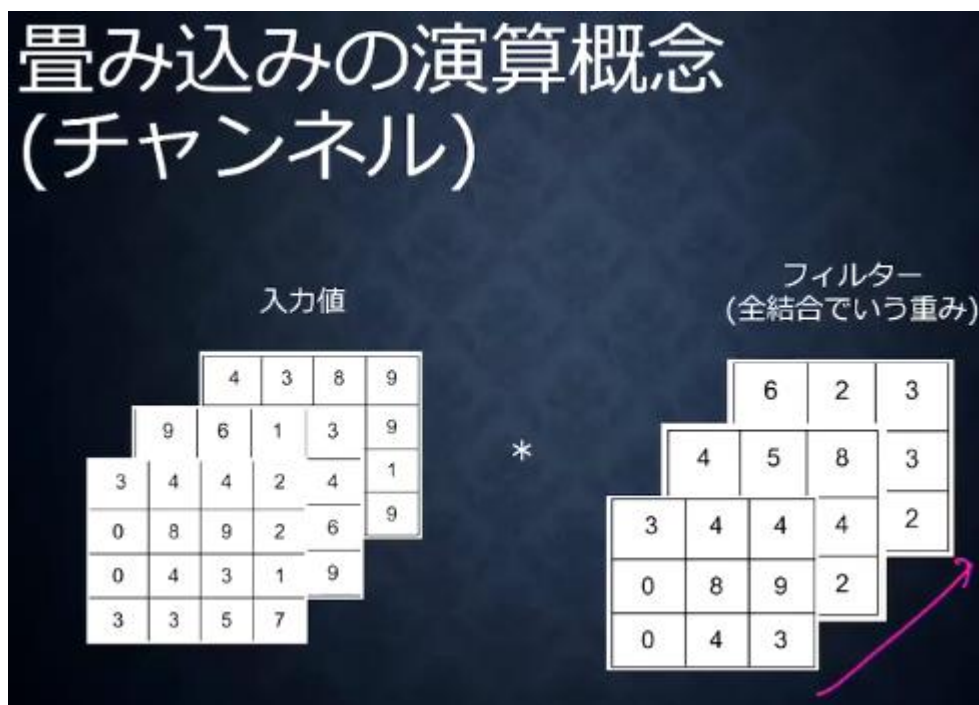


(1) パディングのデータは0とか、隣と同じ数字とかでよい

(2) ストライド



(3) チャンネル



(4) 全結合で画像を学習した際の課題

①全結合層のデメリット

- ・画像の場合、縦、横、チャンネルの3次元データだが、1次元のデータとして処理される。
→RGBの各チャンネル間の関連性が、学習に反映されない。

上記理由により、畳み込み層が生まれた。

4. 7 プーリング層



畳み込みのサイズ計算

公式

$$O_H = \frac{\text{画像の高さ} + 2 \times \text{パディング} - \text{フィルタ高さ}}{\text{ストライド}}$$

$$O_W = \frac{\text{画像の幅} + 2 \times \text{パディング} - \text{フィルタ幅}}{\text{ストライド}}$$

x を行列に変換

```
col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
```

プーリングのサイズに合わせてリサイズ

```
col = col.reshape(-1, self.pool_h*self.pool_w)
```

行ごとに最大値を求める

```
arg_max = np.argmax(col, axis=1)
```

```
out = np.max(col, axis=1)
```

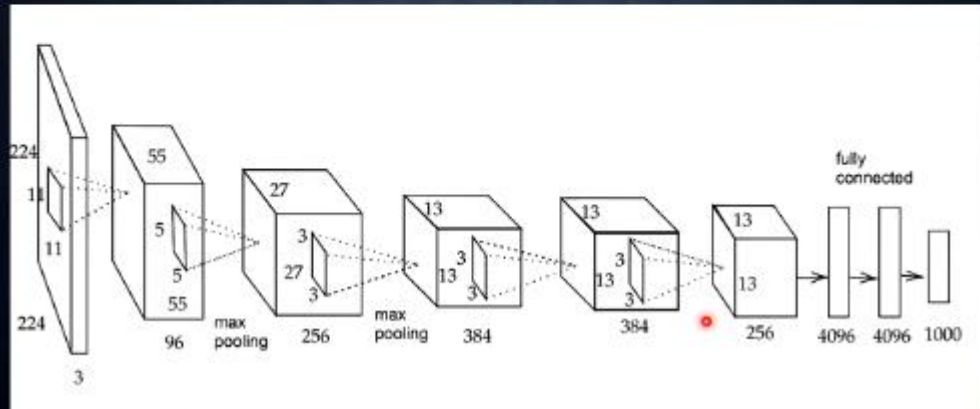
整形

```
out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
```

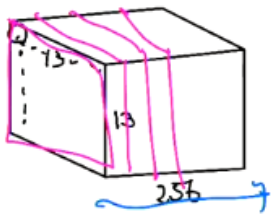

AlexNetのモデル説明

モデルの構造

5層の畳み込み層およびプーリング層など、それに続く3層の全結合層から構成される。



(1) 全結合層への変換



Flatten $\rightarrow [0 \ 0 \ 0 \ \dots \ 000]$
43,264

Global Max Pooling 1番大 $[0 \ 0 \ 0 \ \dots \ 00]$
256

Global Avg Pooling 平均 $[0 \ 0 \ 0 \ \dots \ 00]$
256

過学習を防ぐ施策

- ・サイズ4096の全結合層の出力にドロップアウトを使用している

5 Section5_最新の CNN

5. 1 CNN の変遷

(1) AlexNet(2012)

①ReLU

②LRN(Local Response Normalization)

特徴マップの同一の位置にあり、隣接するチャンネルの出力の値から、自身の出力の値を正規化する方法。

③Overlapping Pooling

Pooling 層をオーバーラップさせる。

④DropOut

隠れ層のニューロンを一定確率で無効化する。

(2) ZfNet(2013)

CNN を可視化して、AlexNet の問題点を明らかにした。

①最初の畳み込み層のフィルタが、大きなカーネルサイズを利用していることから、極端に高周波低周波の情報を取得するフィルタとなっており、それらの間の周波数成分を取得するフィルタがほとんどなかった。

②2 層目の特徴マップにおいてエイリアシングが発生している。

③解決方法

- ・最初の畳み込み層のフィルタサイズを 11 から 7 に縮小する。
- ・ストライドを 4 から 2 に縮小する。

(3) GoogleNet(2014)

①Inception モジュール

複数の畳み込み層や Pooling 層から構成される Inception モジュールと呼ばれる小さなネットワークを定義し、これを通常の畳み込み層のように重ねていくことにより 1 つの大きな CNN を作り上げている。

②Global Average Pooling

③Auxiliary Loss

④Inception-vX

(4) VGGNet(2014)

シンプルなモデルアーキテクチャや学習モデルが配布されている。

① 3×3 の畳み込みを利用する。

②同一チャンネル数の畳み込み層をいくつか重ねた後に、max pooling により特徴マップを半分に縮小する。

③max pooling の後の畳み込み層の出力チャンネル数を 2 倍に増加させる。

(5) ResNet(2015)

ネットワークを深くする(VGGNet)ことは表現能力を向上させ、認識精度を改善させるが、あまりにも深いネットワークは効率的な学習が困難であった。

通常のネットワークのように、何かしらの処理ブロックによる変換を単純に次の層に渡していくのではなく、そのブロックへの入力をショートカットし、次の層に渡していく。

①Residual モジュール

上記ショートカットの名称

②Batch Normalization

内部共変量シフトを正規化し、なるべく各レイヤが独立して学習が行えるようにすること。

(6) SENet(2017)

特徴マップをチャンネル毎に適応的に重み付けする Attention の構造を導入

5. 2 最新の CNN 改良手法

①. Residual モジュールの改良

- ResNet は residual モジュールを重ねていくだけというシンプルな設計でありながら、高精度な認識を実現できることから、デファクトスタンダードなモデルとなった。これに対し、residual モジュール内の構成要素を最適化することで、性能改善を図る手法が複数提案されている。
- WideResNet
- PyramidNet

②. 独自モジュールの使用

- ResNetXt
- Xception
- Separatable 畳み込み
- Xception モジュール

③. 独自マクロアーキテクチャの利用

- RoR
- FractalNet
- DenseNet

④. 正則化

- Stochastic Depth
- Swapout
- Shake-Shake Regularization
- ShakeDrop
- Cutout/Randam Erasing
- mixup

⑤. 高速化を意識したアーキテクチャ

- SqueezeNet
- MobileNet

⑥. アーキテクチャの自動設計