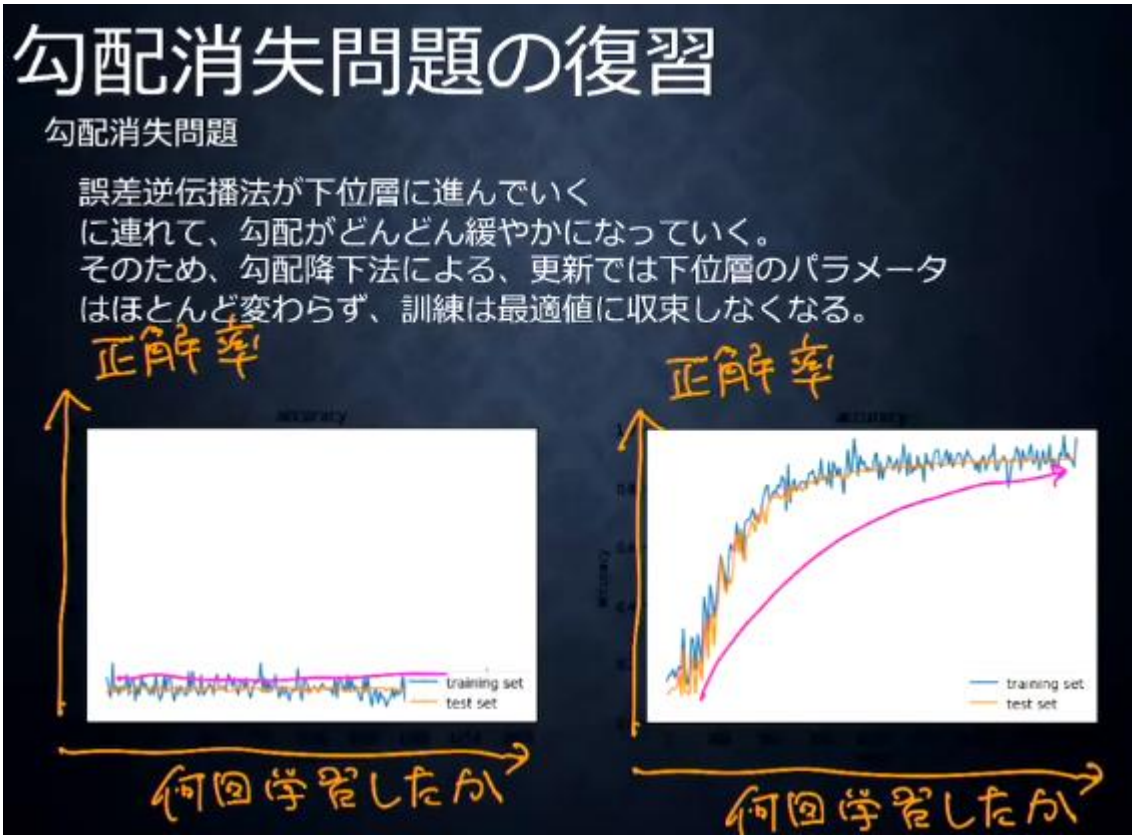
目　次

内容

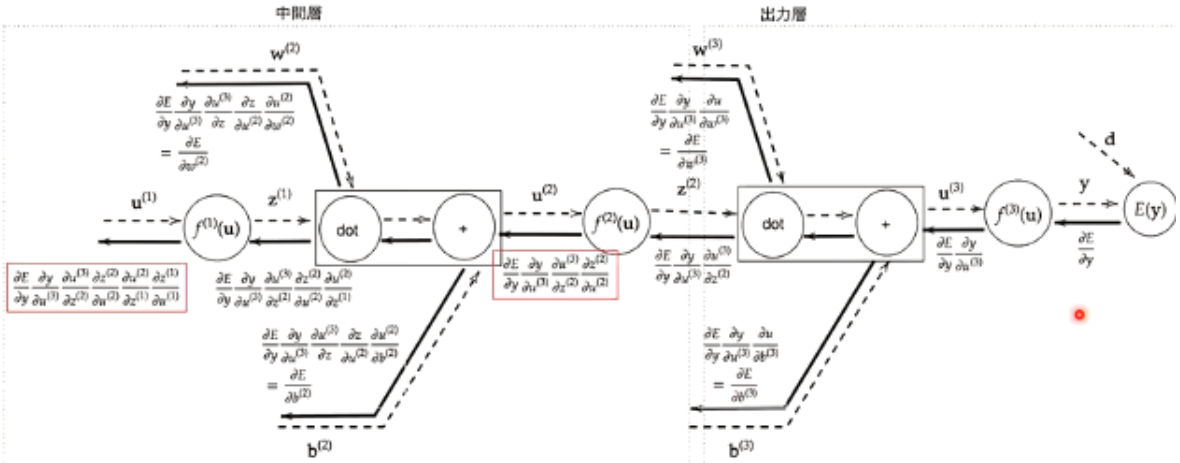
1　Section1_勾配消失問題 2
1．1　全体像 2
1．2　活性化関数 4
1．3　初期値の設定方法 4
1．4　バッチ正規化 6
2　Section2_学習率最適化手法 8
2．1　全体像 8
2．2　モメンタム 9
2．3　Adagrad 10
2．4　RMSProp 11
2．5　Adam 12
3　Section3_過学習 13
3．1 全体像 13
3．2　正規化手法１ 14
3．3　正規化手法２ 15
3．4　正規化手法３ 16
3．5　正規化手法４ 18
4　Section4_畳み込みニューラルネットワークの概念 19
4．1　構造１ 19
4．2　構造２ 20
4．3　全体像 21
4．4　畳み込み層（バイアス） 23
4．5　畳み込み層（パディング・ストライド） 24
4．7　プーリング層 26
4．8　Alexnet 27
5　Section5_最新の CNN 28
5．1　CNN の変遷 28
5．2　最新の CNN 改良手法 30

1　Section1_勾配消失問題

1．1　全体像



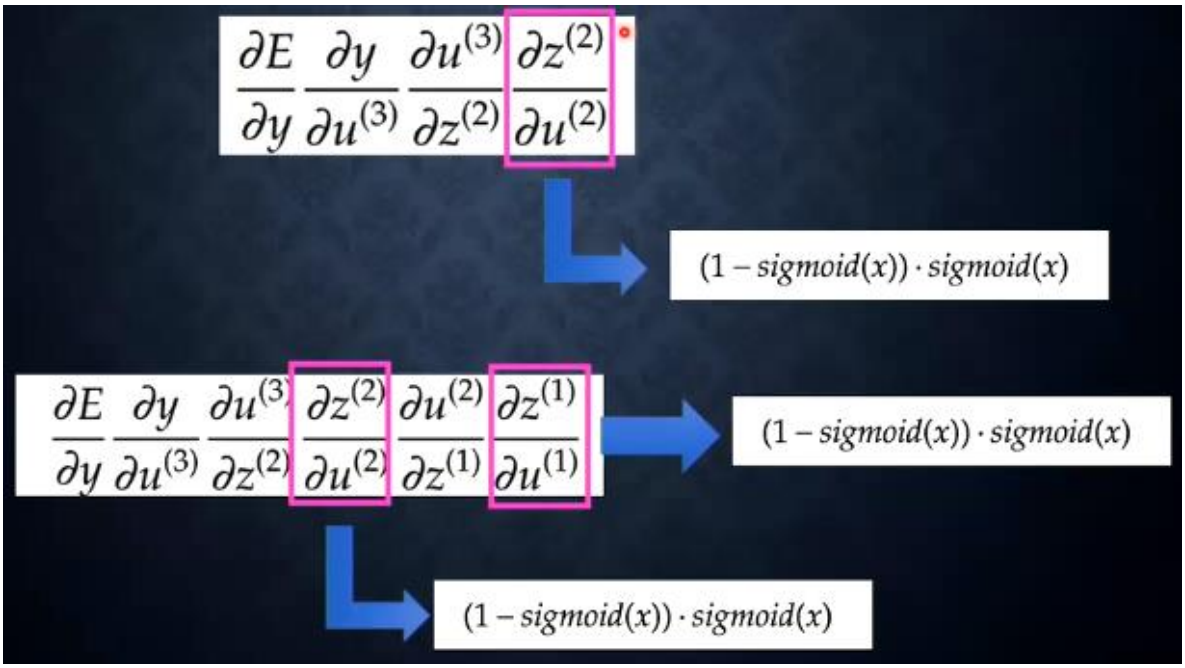
1）勾配消失問題のビジョン



微分値は０～１の価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＜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

①．ReLU 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

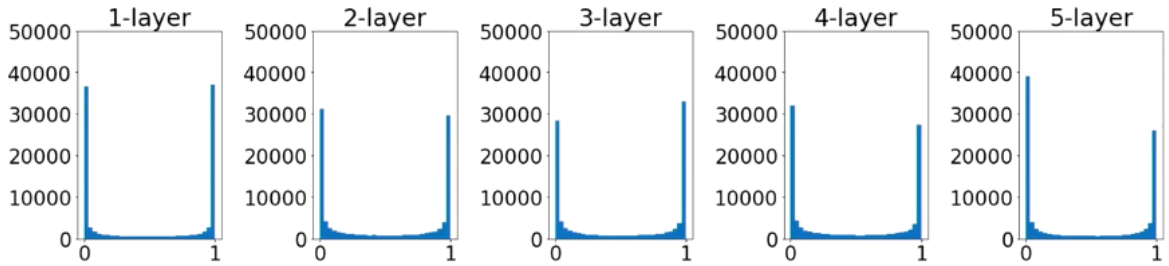
標準正規分布に基づいて初期化する（平均が０、分散が１）

```
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

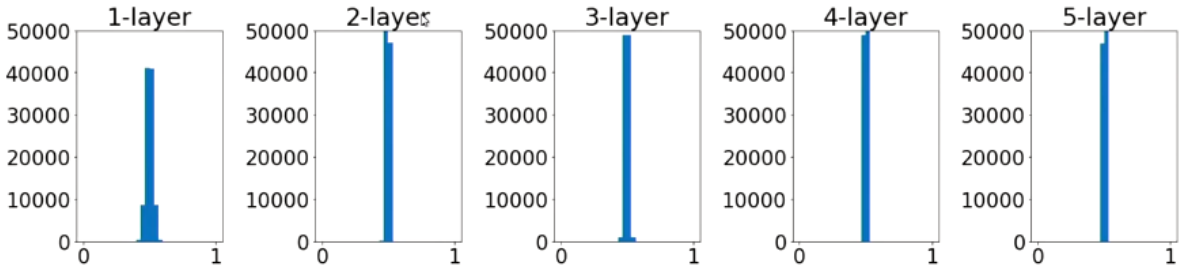
１）標準正規分布で重み付け



１か０を出力

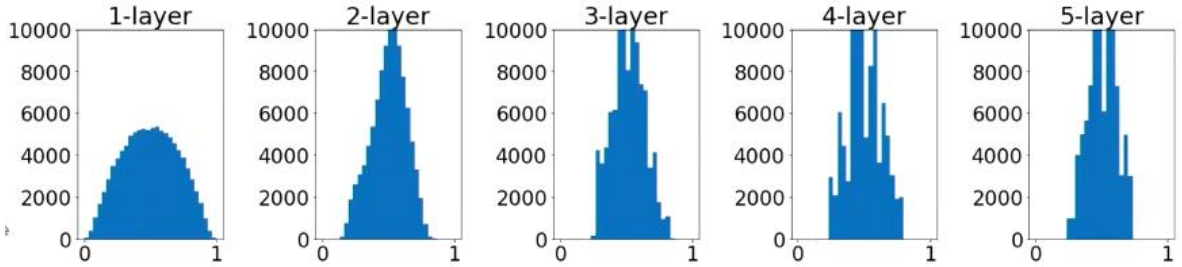
逆伝播に向かない

２）適当に小さい値で割る



０．５による

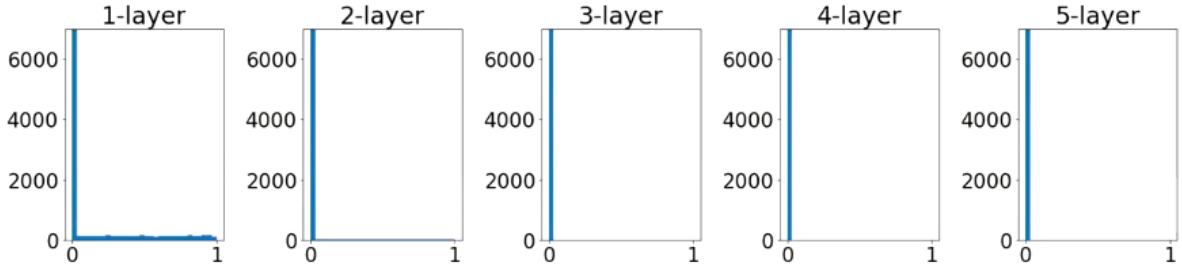
3）Xavier の初期化



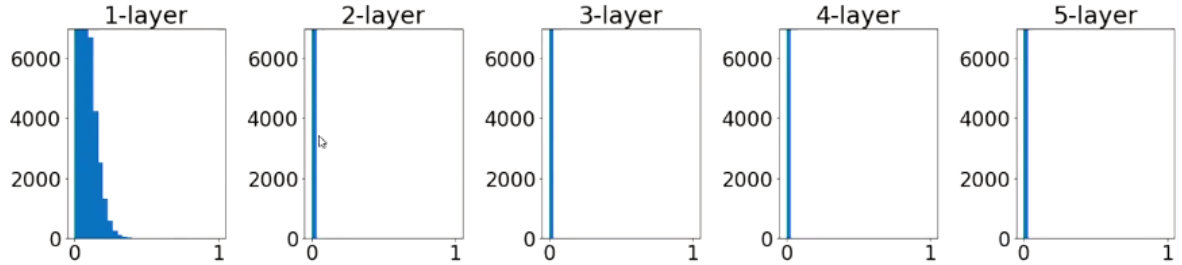
S 字カーブの関数によく働く

4）初期設定値を He 初期化（ReLU 関数）

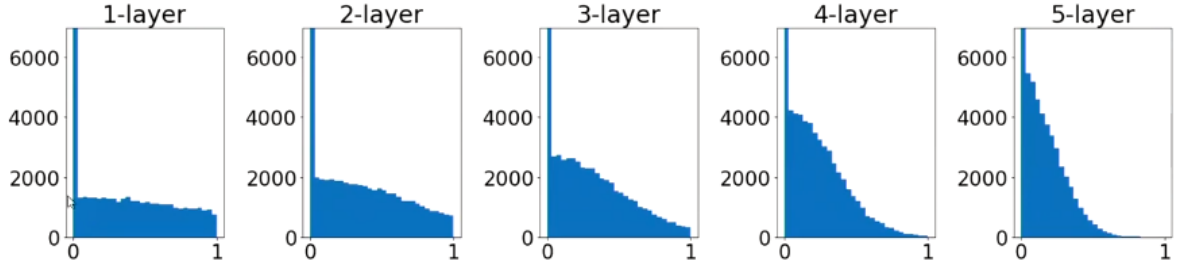
①. 標準偏差



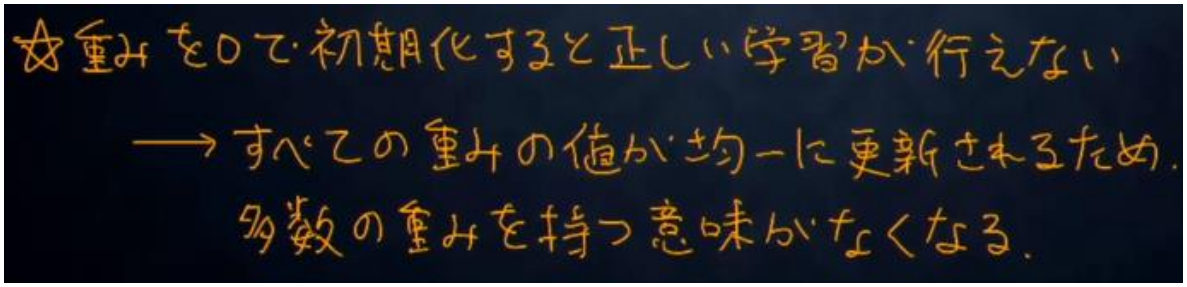
②. 標準偏差を値を小さくした



③. He 初期化



5）重みの初期値を 0 にする



１．４　バッチ正規化

１）バッチ正規化とは

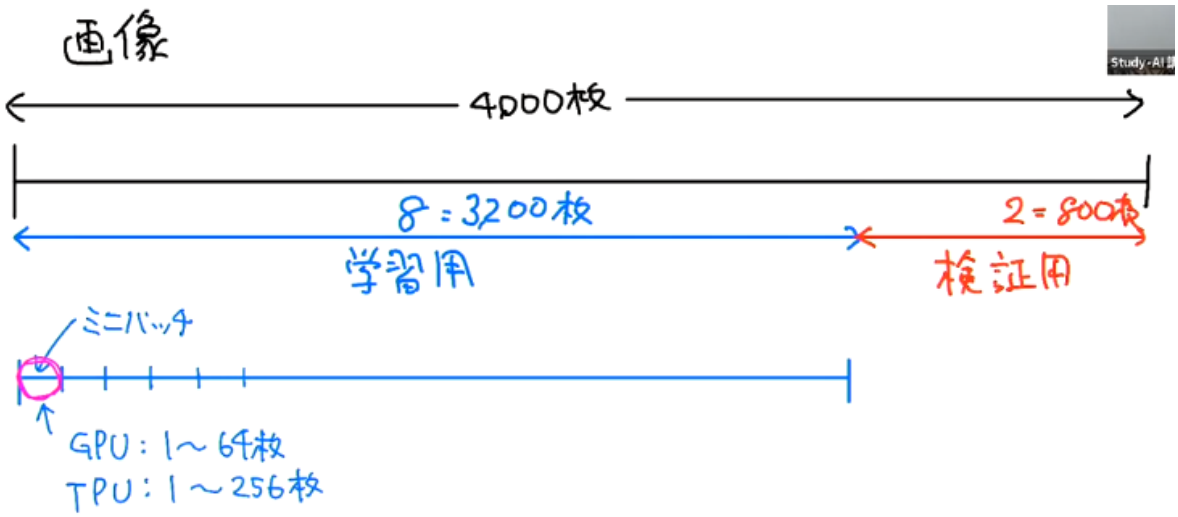
ミニバッチ単位で、入力値のデータの偏りを抑制する手法

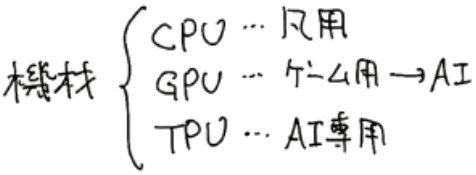
２）バッチ正規化の使いどころとは

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 又は z





・中間層の重みの更新がうまくいく

・過学習が起こりにくい

・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

ミニバッチの分布

ミニバッチの正規化

変倍移動

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. $y_{ni} = \gamma x_{ni} + \beta$

４）処理及び記号の説明

μ_t：ミニバッチt全体の平均

$\sigma_t{}^2$：ミニバッチt全体の標準偏差

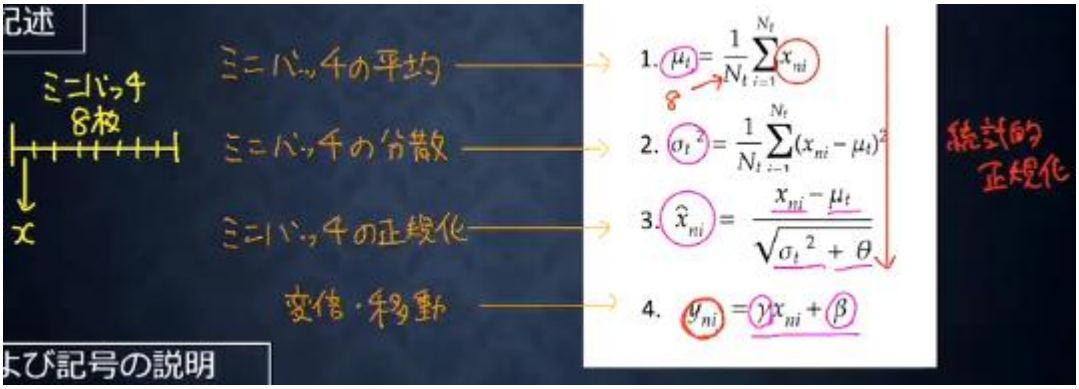
N_t：ミニバッチのインデックス

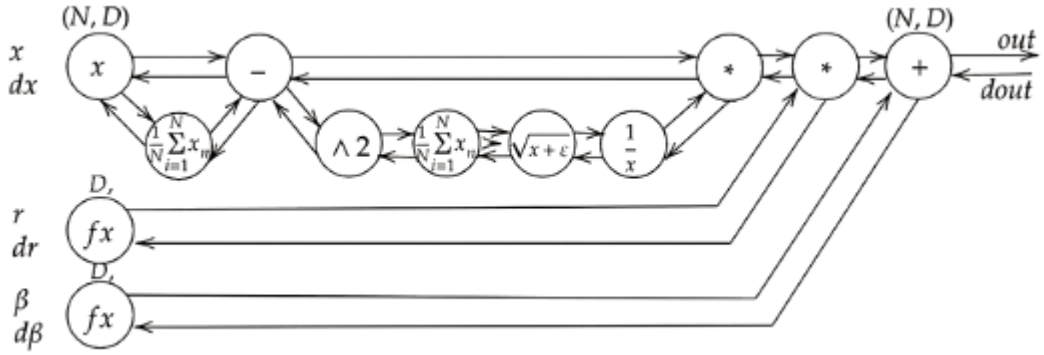
$\hat{x}_{ni}$：0に値を近づける計算(0を中心とするセンタリング)と正規化を施した値

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)

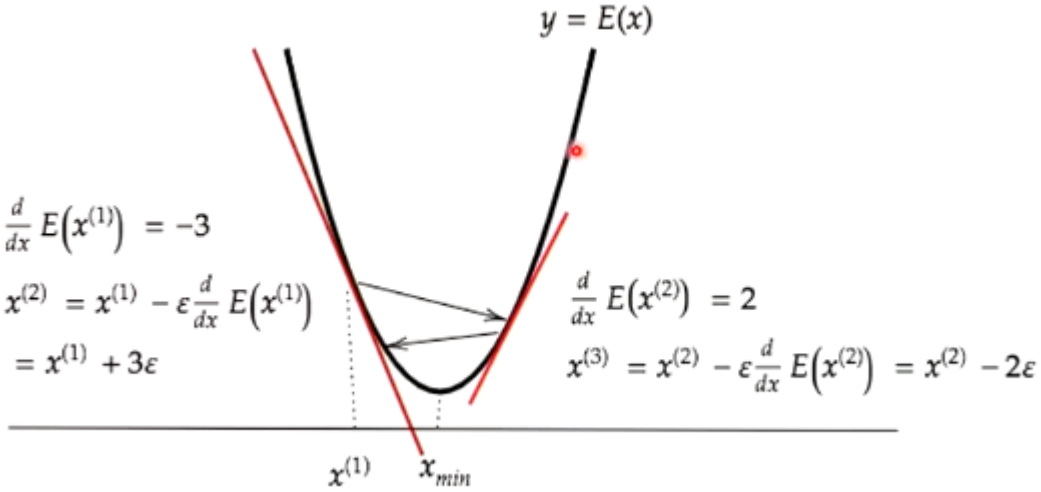

記述
ミニバッチ
8枚
x
ミニバッチの平均
ミニバッチの分散
ミニバッチの正規化
変倍・移動
1. $\mu_t = \frac{1}{N_t}\sum_{i=1}^{N_t} x_{ni}$
2. $\sigma_t^2 = \frac{1}{N_t}\sum_{i=1}^{N_t}(x_{ni}-\mu_t)^2$
3. $\hat{x}_{ni} = \frac{x_{ni}-\mu_t}{\sqrt{\sigma_t^2+\theta}}$
4. $y_{ni} = \gamma x_{ni} + \beta$
8
統計的正規化
よび記号の説明


(N, D)
x
dx
−
∗
∗
+
(N, D)
out
dout
∧2
$\frac{1}{N}\sum_{i=1}^{N} x_n$
$\sqrt{x+\varepsilon}$
$\frac{1}{x}$
D,
r
dr
fx
D,
β
dβ
fx

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

・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

・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。

・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

・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

・パラメータごろ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

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

```
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

勾配降下法

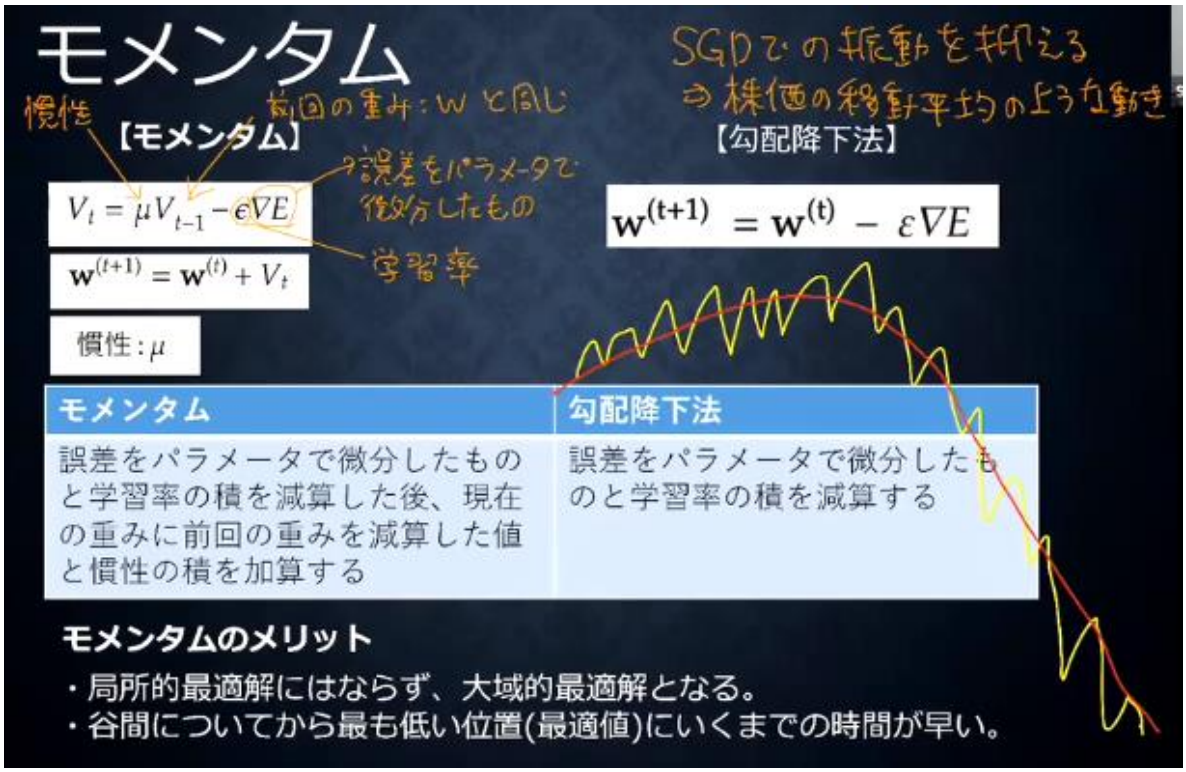
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E$$

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

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)
```

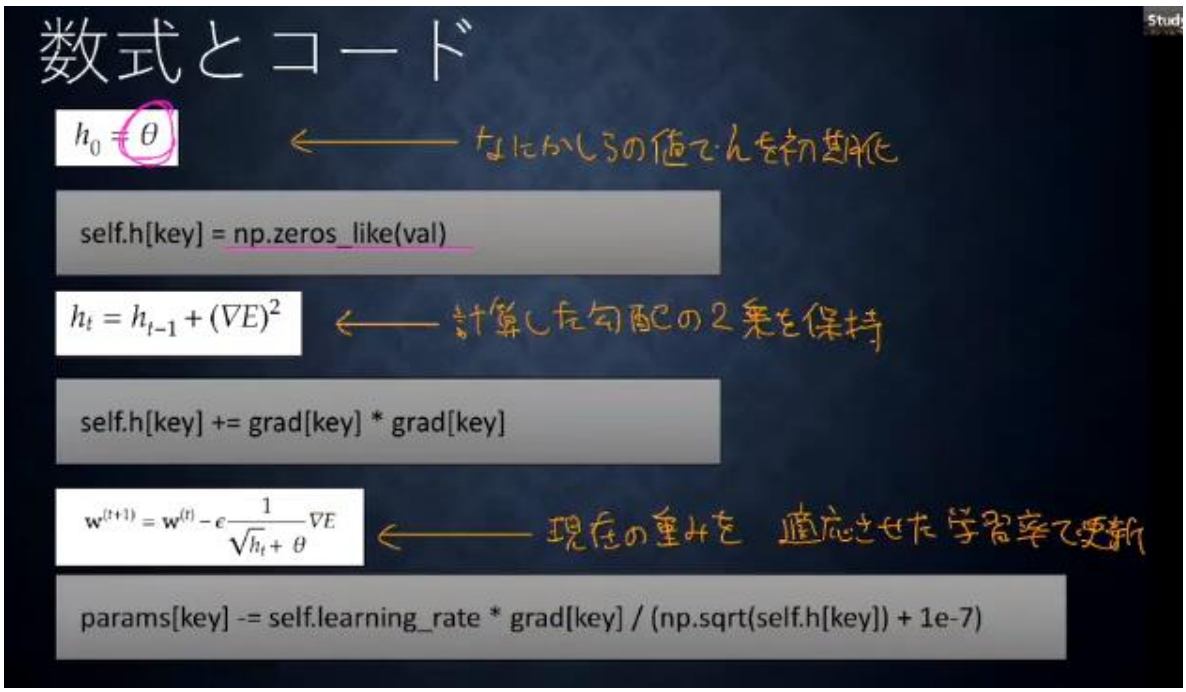
誤差をパラメータで微分したものと
再定義した学習率の積を減算する

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。



２．４　RMSProp

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

α ： 0 ～ 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

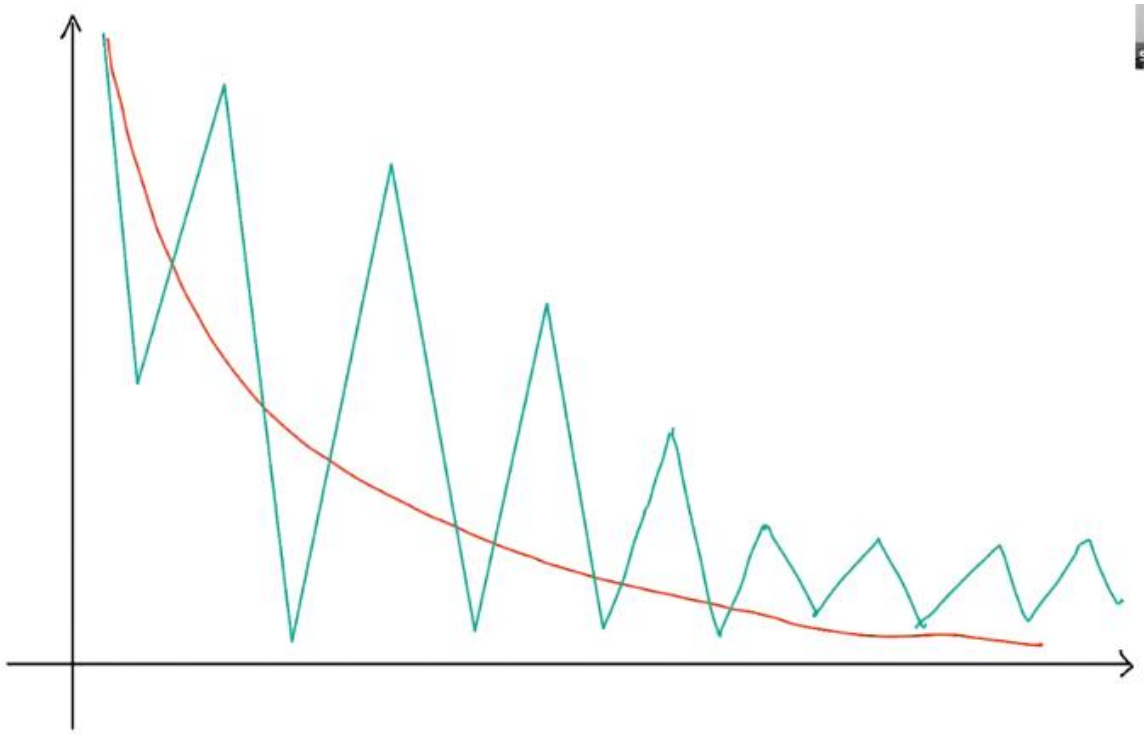
①．モメンタムの、過去の勾配の指数関数的減衰平均

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。



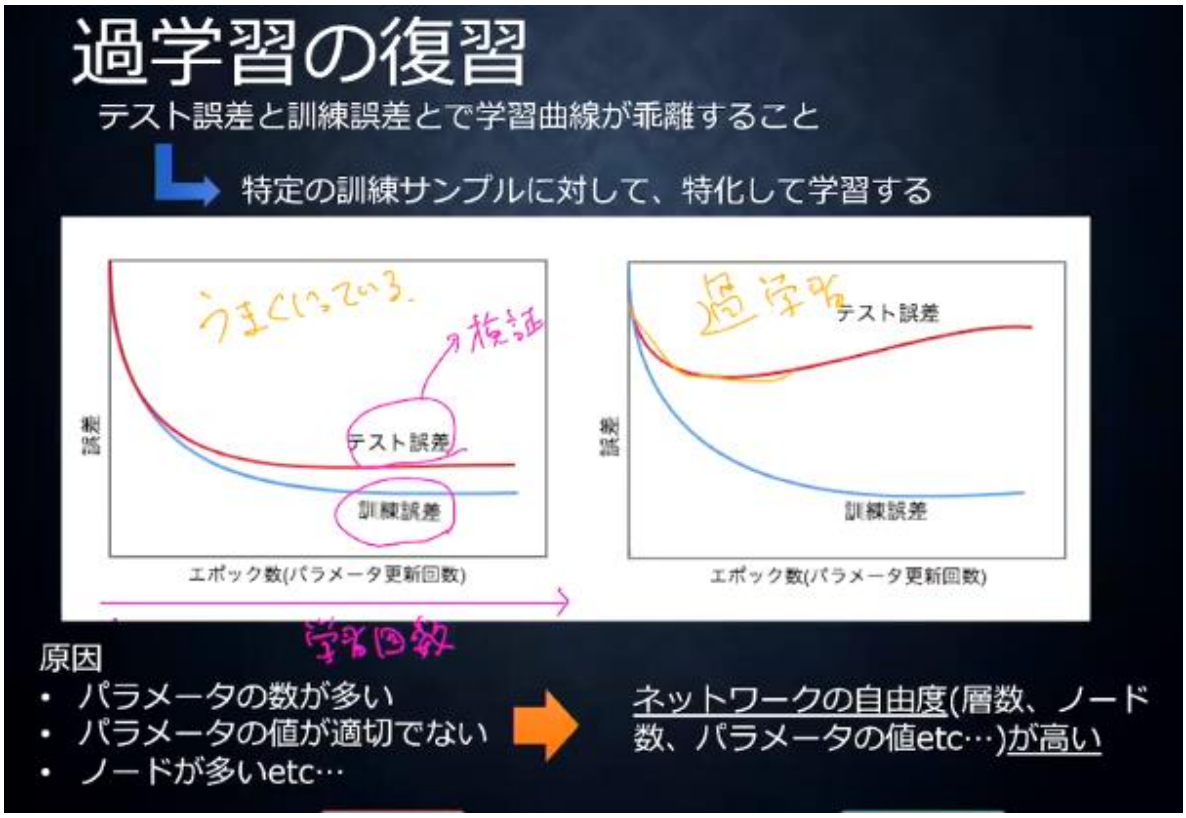
３　Section3_過学習

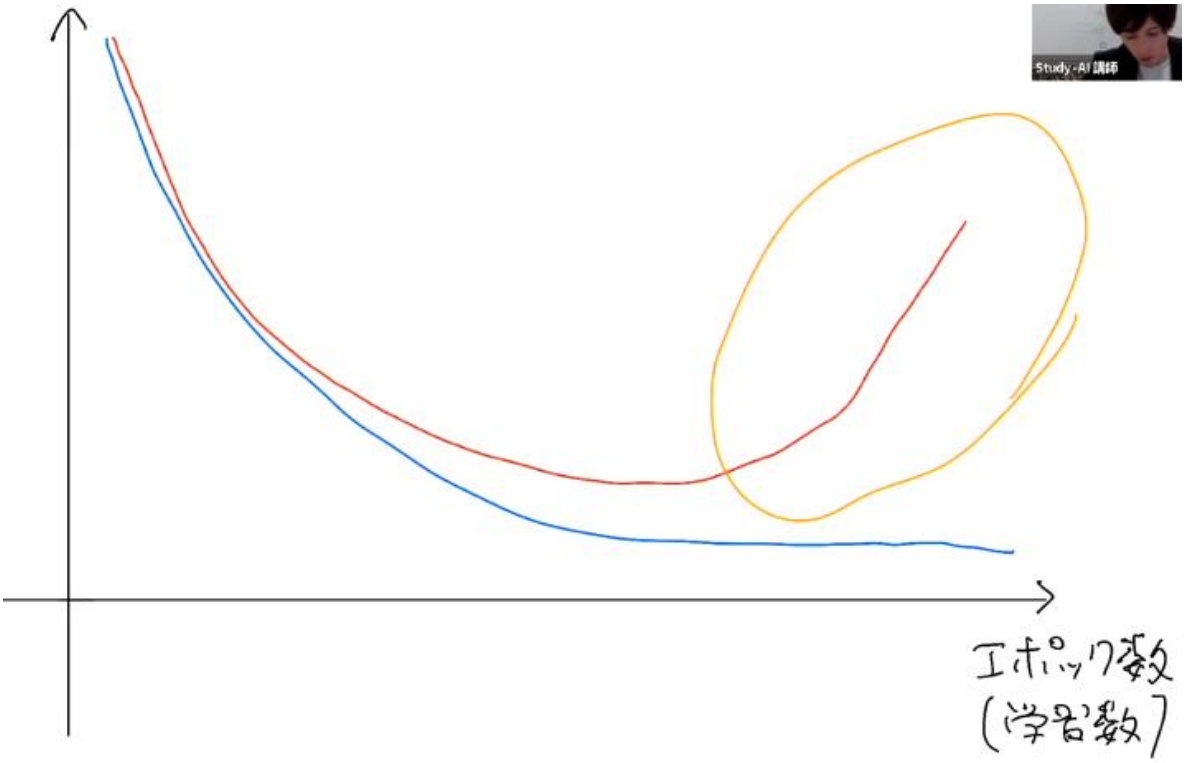
３．１全体像

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

①．L1 正則化、L2 正則化

②．ドロップアウト

３）Weight decary（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すし、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

$$E_n(\mathbf{w}) + \frac{1}{p}\lambda \parallel x \parallel_p$$

：誤差関数に、ｐノルムを加える（距離）

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

$$\parallel x \parallel_p = \left(\mid x_1 \mid^p + \ldots + \mid x_n \mid^p \right)^{\frac{1}{p}}$$

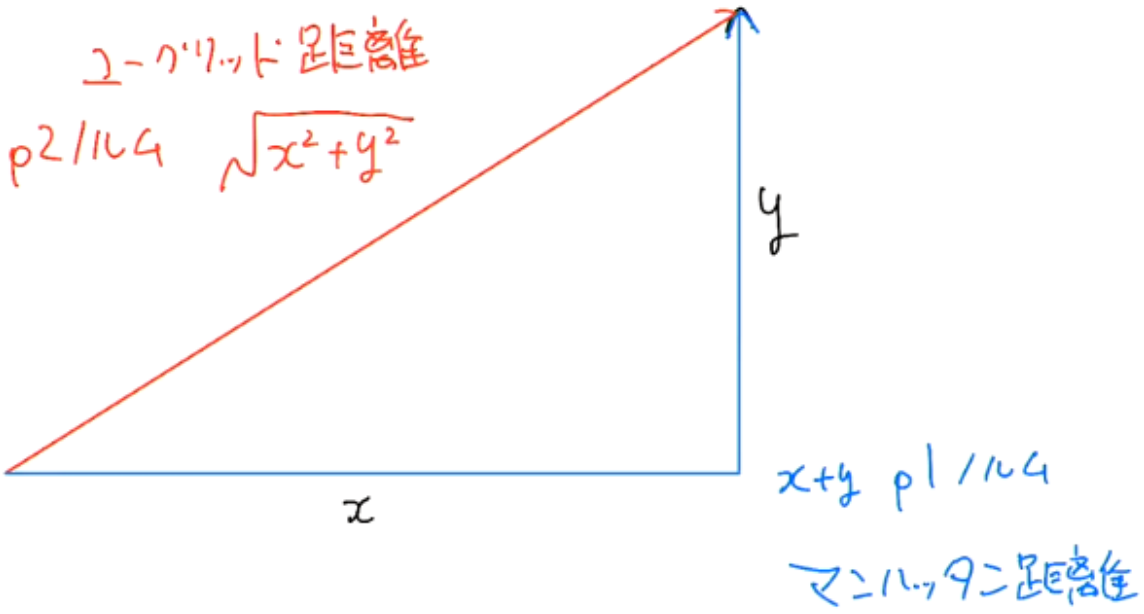
：ｐノルムの計算

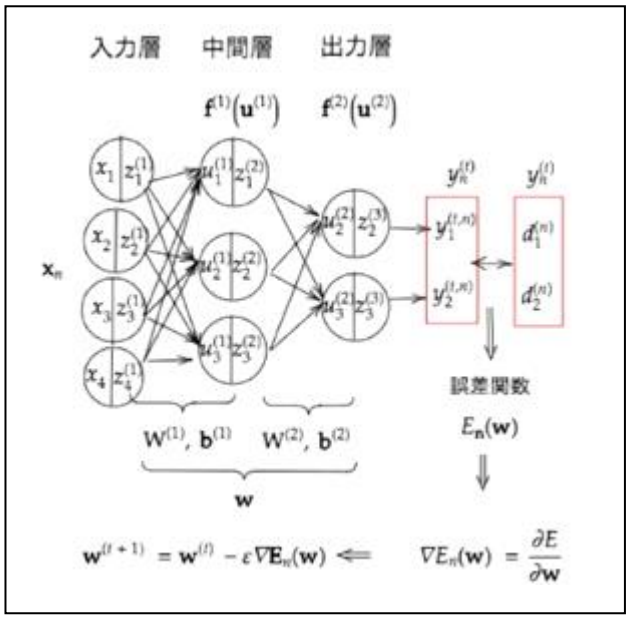
```
weight_decay += weight_decay_lambda
*np.sum(np.abs(network.params['W' + str(idx)]))

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





$$\| \mathbf{W}^{(1)} \|_p = \left(| \mathbf{W}_1^{(1)} |^p + \ldots + | \mathbf{W}_n^{(1)} |^p \right)^{\frac{1}{p}}$$

$$\| \mathbf{W}^{(2)} \|_p = \left(| \mathbf{W}_1^{(2)} |^p + \ldots + | \mathbf{W}_n^{(2)} |^p \right)^{\frac{1}{p}}$$

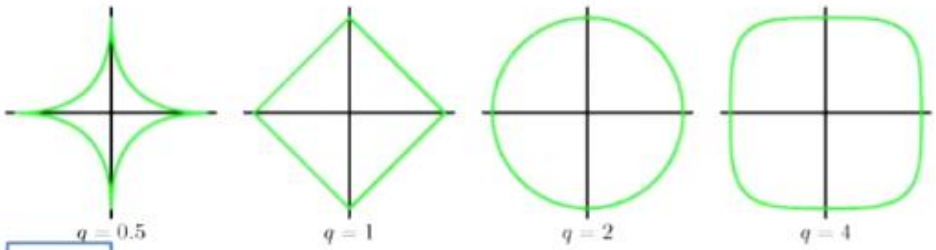
$$\| x \|_p = \| \mathbf{W}^{(1)} \|_p + \| \mathbf{W}^{(2)} \|_p$$

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

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 $\sum_{j=1}^{M}\left|w_j\right|^q$

Rで描いた三次元グラフ

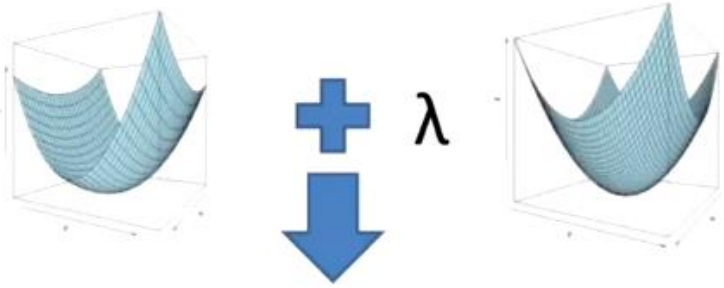


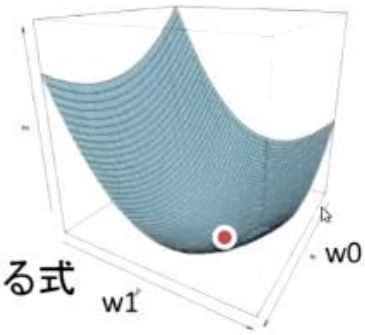
補足　PRML

等高線の形（三次元グラフを横で切った形）

q=2のときの最小化

誤差関数　　正則化項

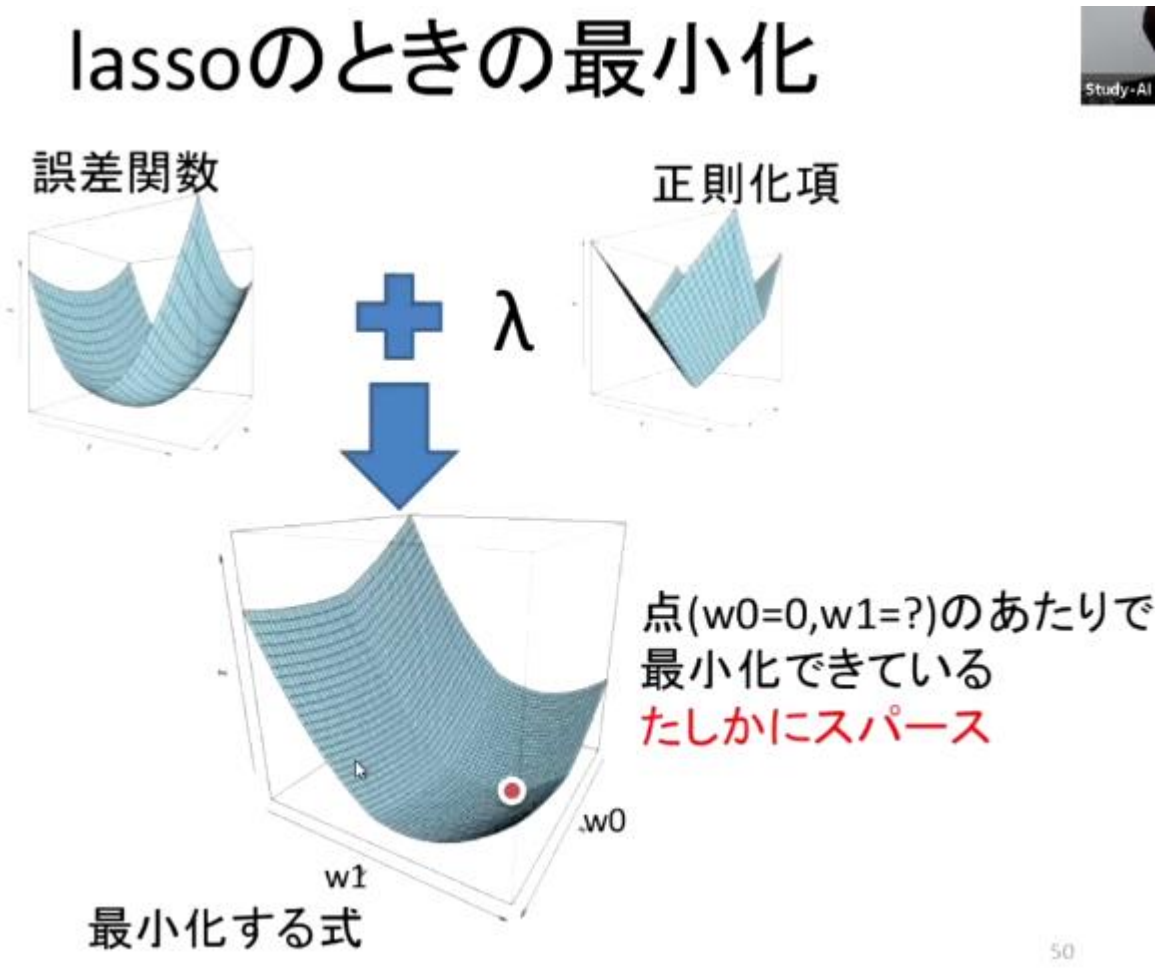




最小化する式

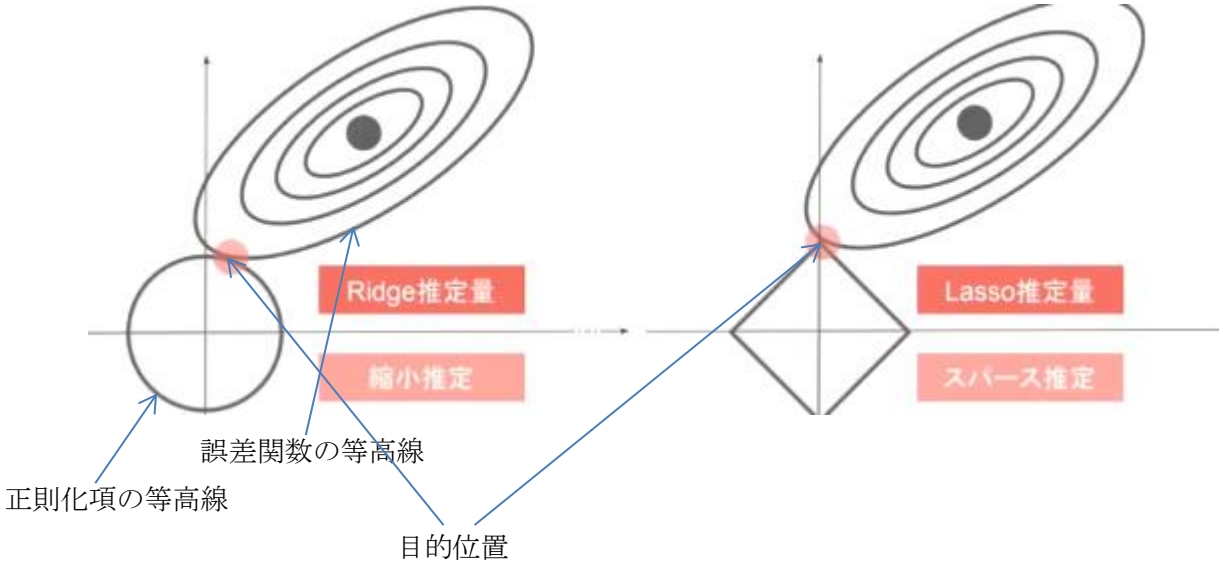
点(w0=?,w1=?)のあたりで
最適解になっている
どうみてもスパースでない

補足



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

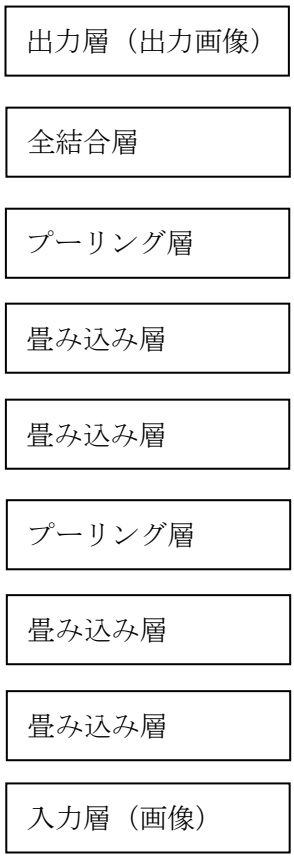


４　Section4_畳み込みニューラルネットワークの概念

４．１　構造１

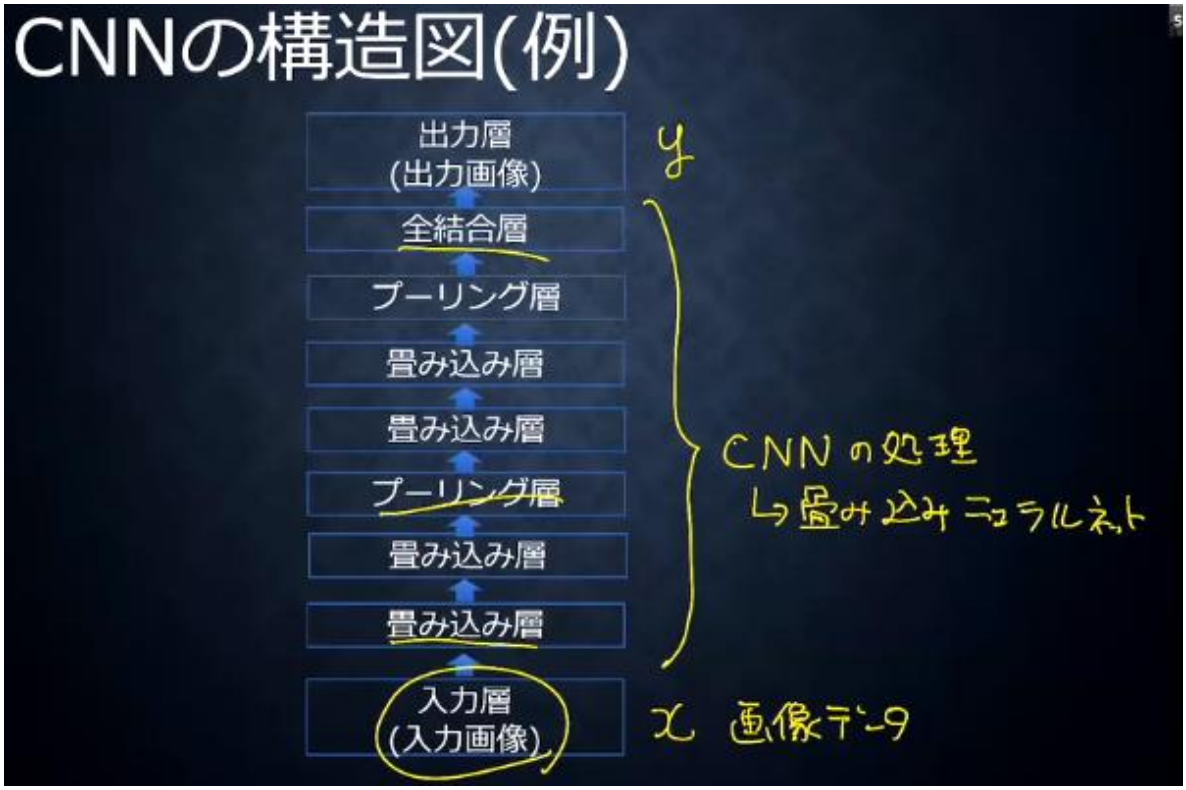
１）CNN の構造図（例）

CNN は次元間でつながりのあるデータを扱える

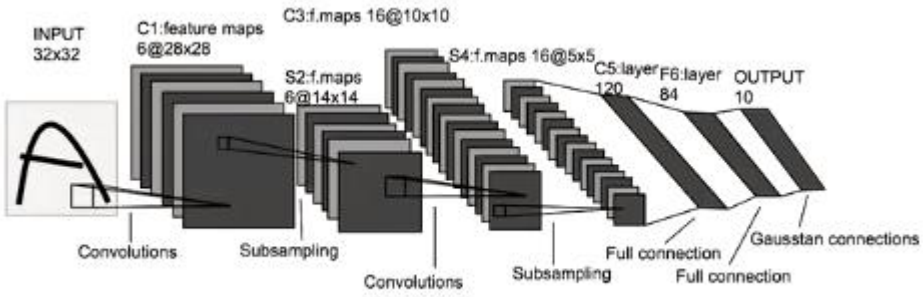


	1 次元	2 次元	3 次元
単一チャネル	音声（時刻、強度）	フーリエ変換した音声 （時刻、周波数、強度）	CT スキャン画像
複数チャネル	アニメのスケルトン （時刻、腕の価、膝の価、..）	カラー画像 （ｘ、ｙ、(R,G,B)）	動画 （時刻、ｘ、ｙ、(R,G,B)）

４．２　構造２



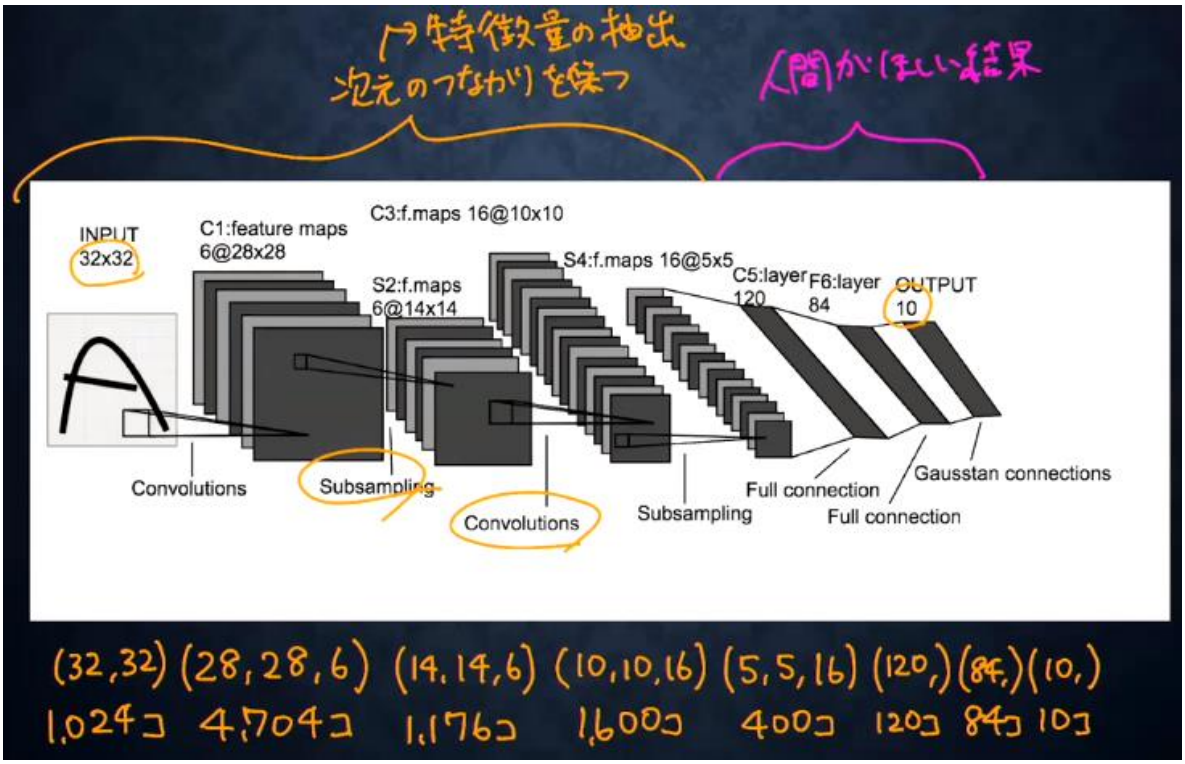
1）LeNet の構造図



(32,329　(28,28,6)　(14,14,6) (10,10,16) (5,5,16) (120,) (84,) (10,)

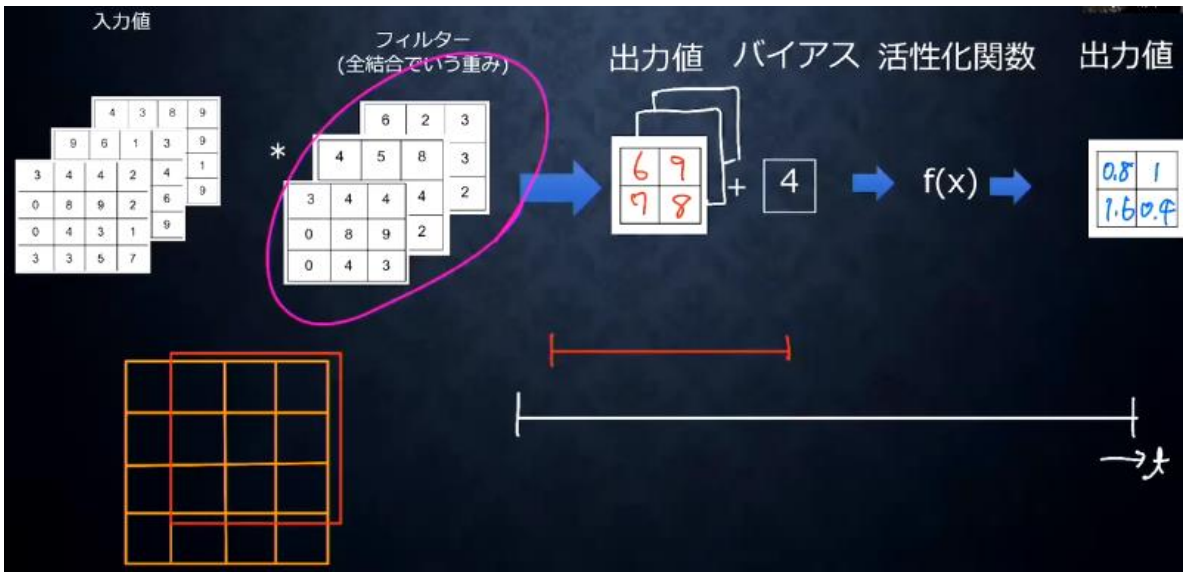
1024　　4704　　1176　　1600　　400

全結合層

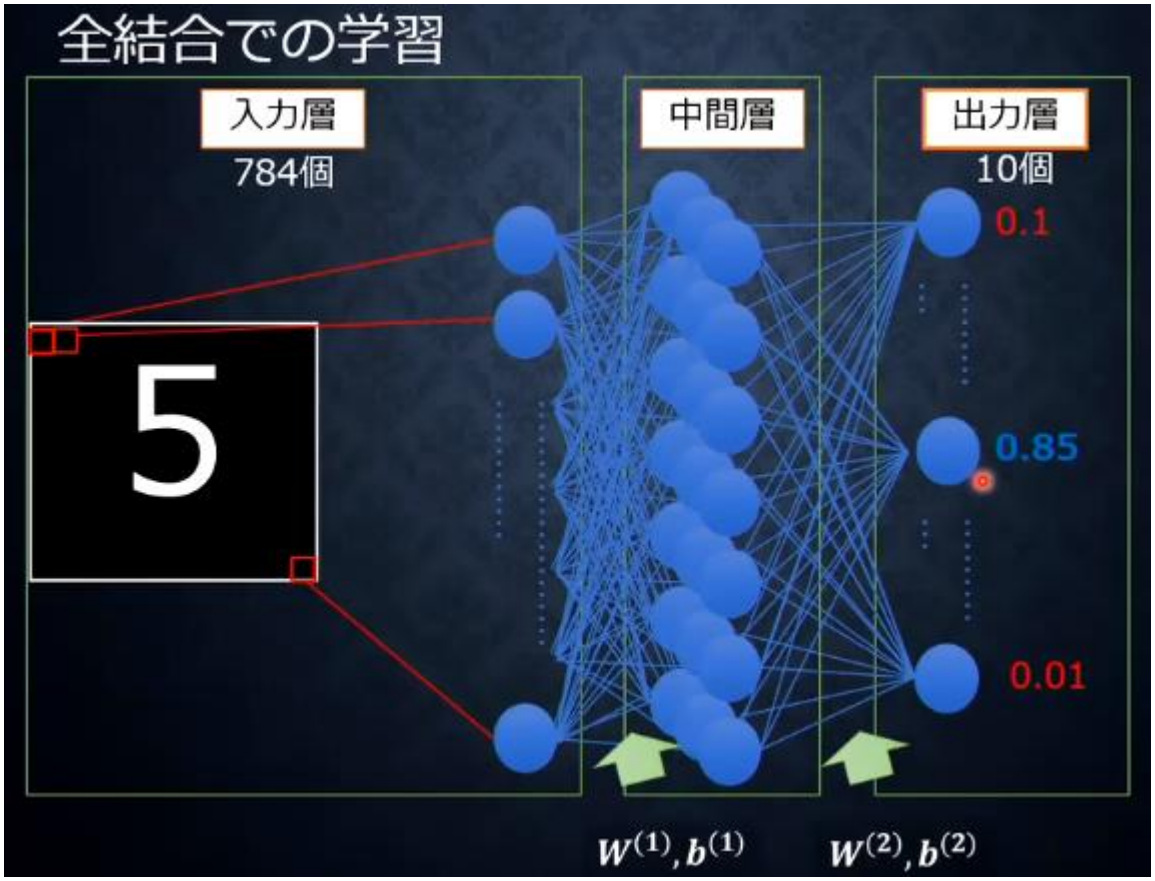


４．３　全体像

１）畳み込み層



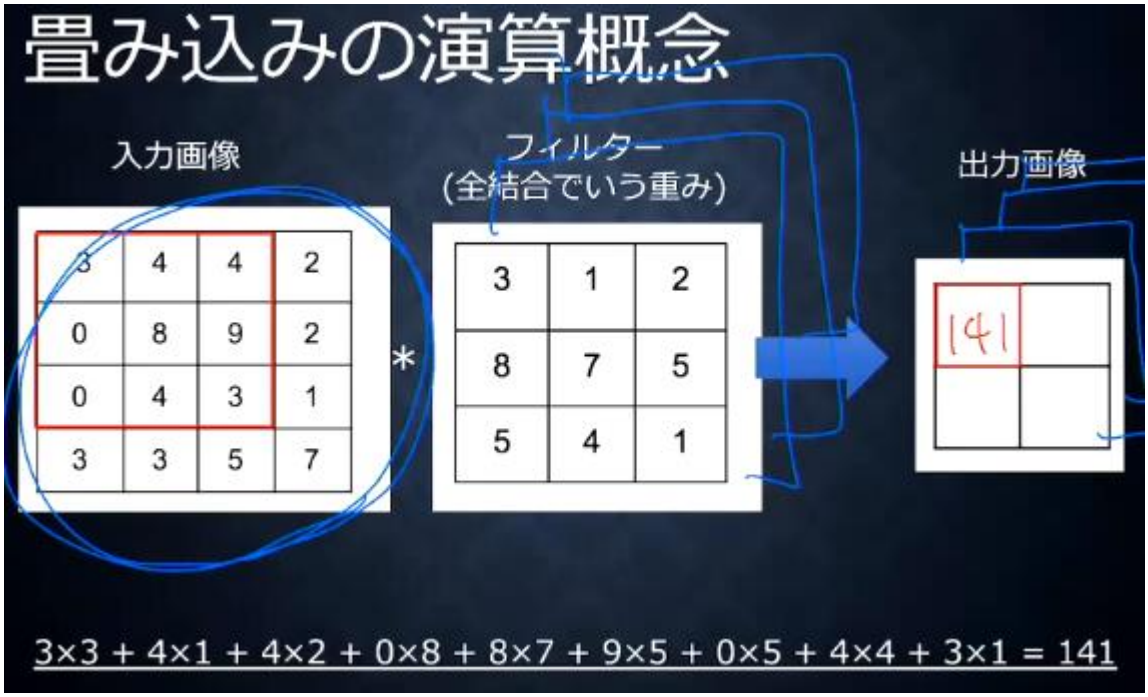
次元間の繋がりが保たれている。

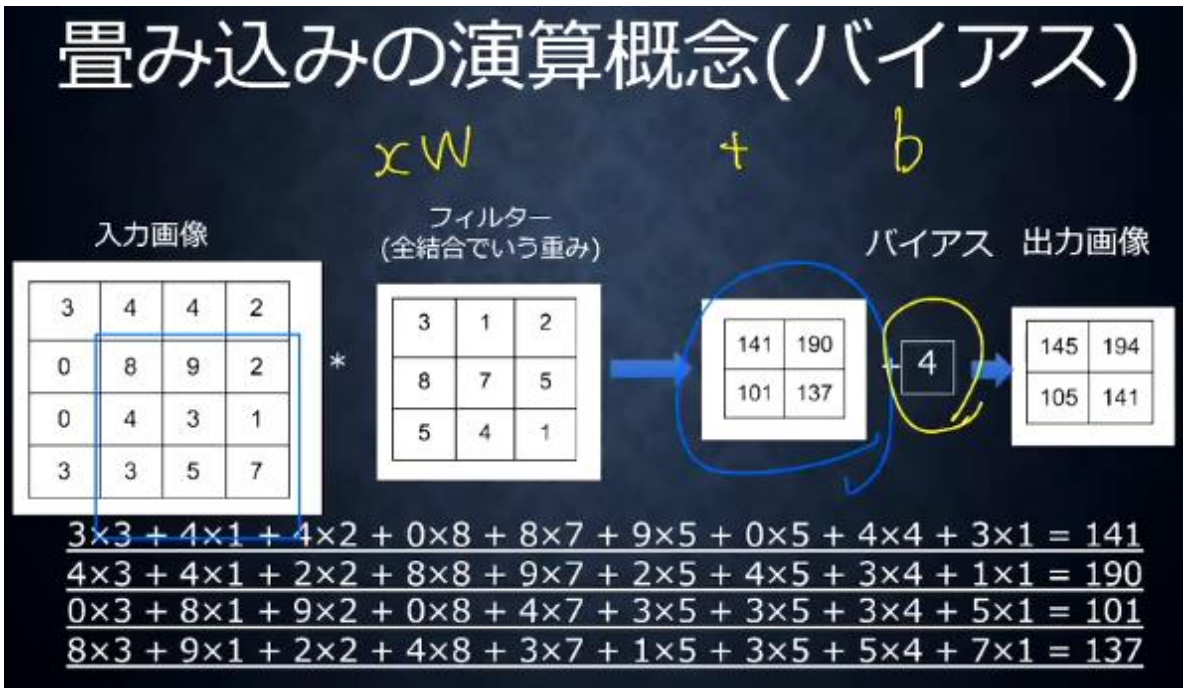

全結合での学習
入力層
784個
中間層
出力層
10個
0.1
0.85
0.01
5
$W^{(1)}, b^{(1)}$
$W^{(2)}, b^{(2)}$

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。



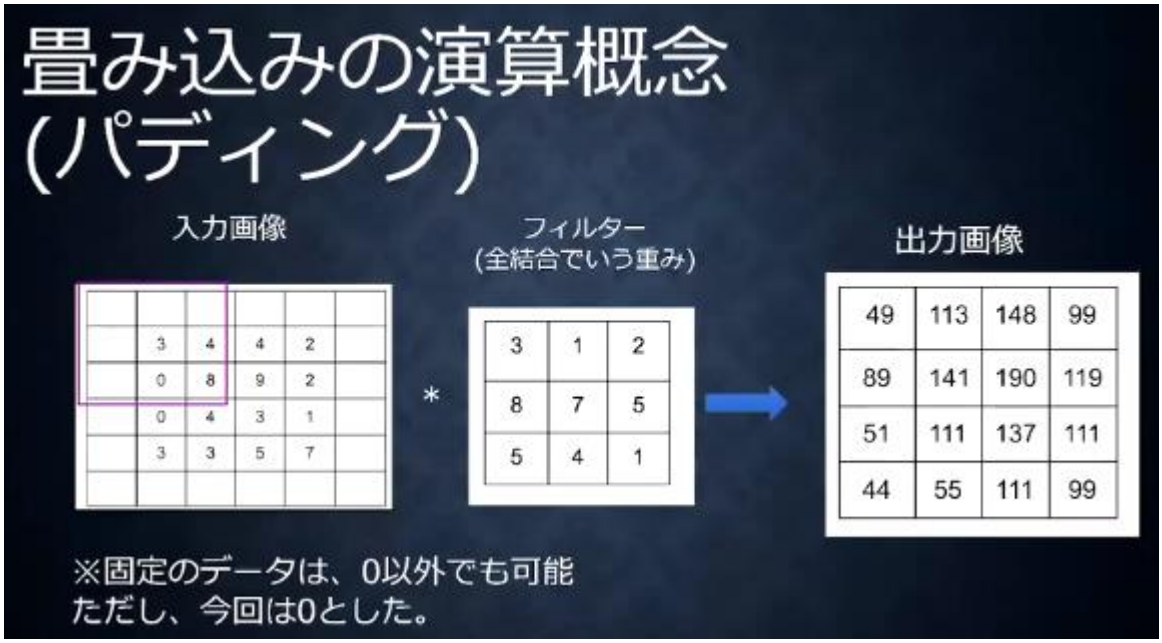


３）パディング

４×４の画像を３×３のフィルターを通すと、２×２となる。

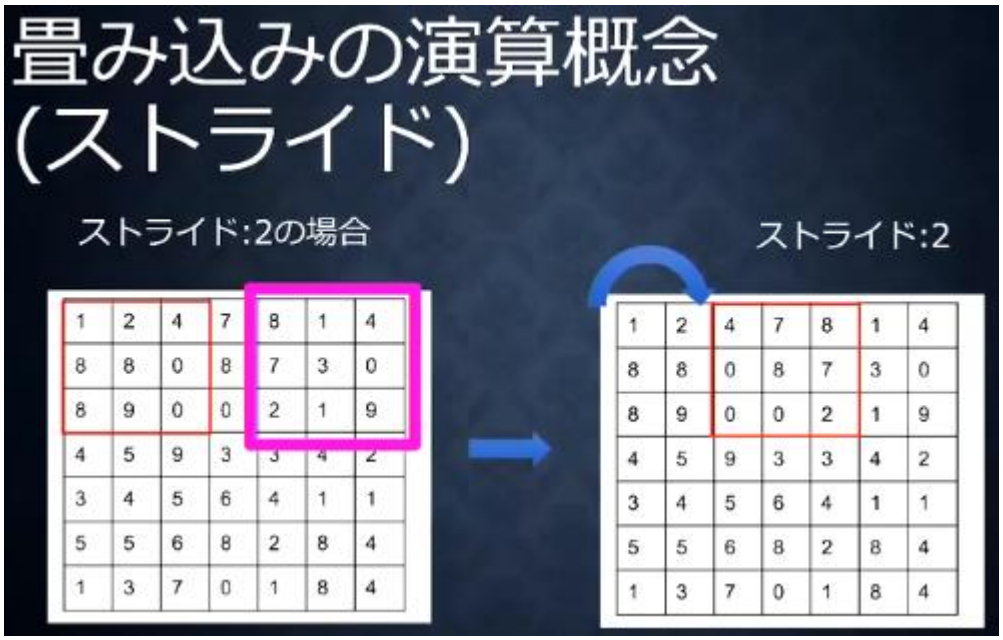
このため、パディングを行い、出力画像の画素数を減らないようにする。

４．５　畳み込み層（パディング・ストライド）

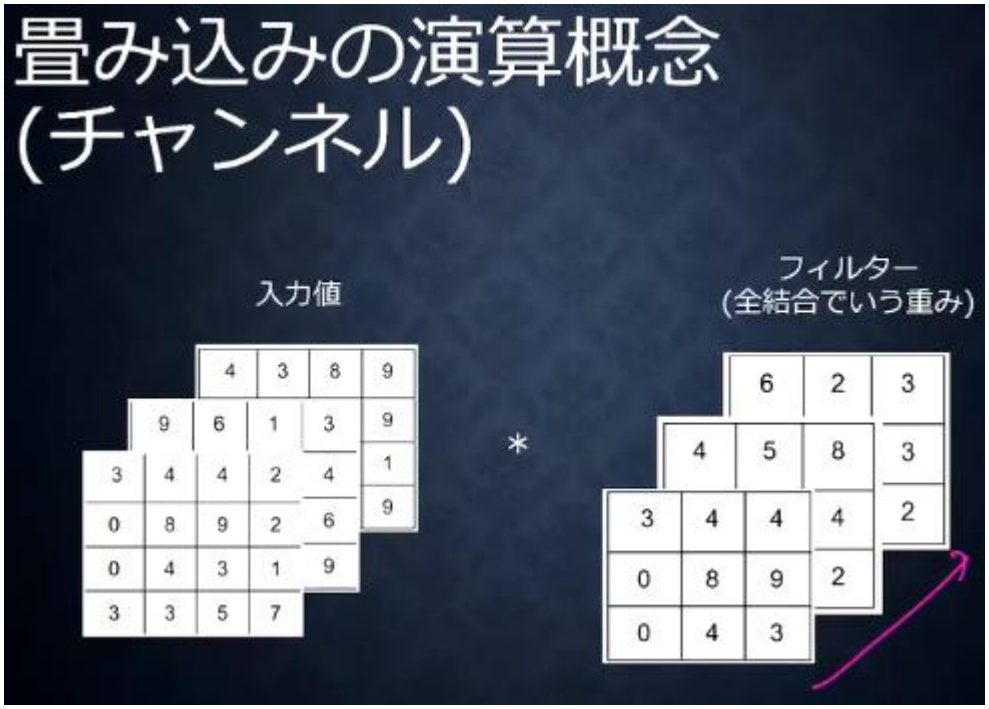


1）パディングのデータは０とか、隣と同じ数字とか

2）ストライド



3）チャネル



4）全結合で画像を学習した際の課題

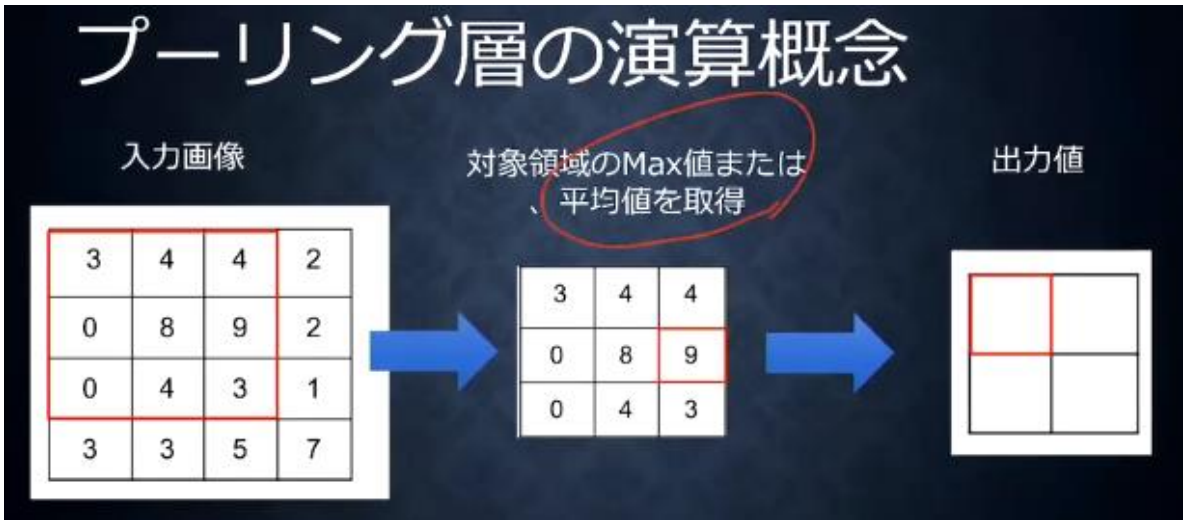
①. 全結合層のデメリット

・画像の場合、縦、横、チェネルの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

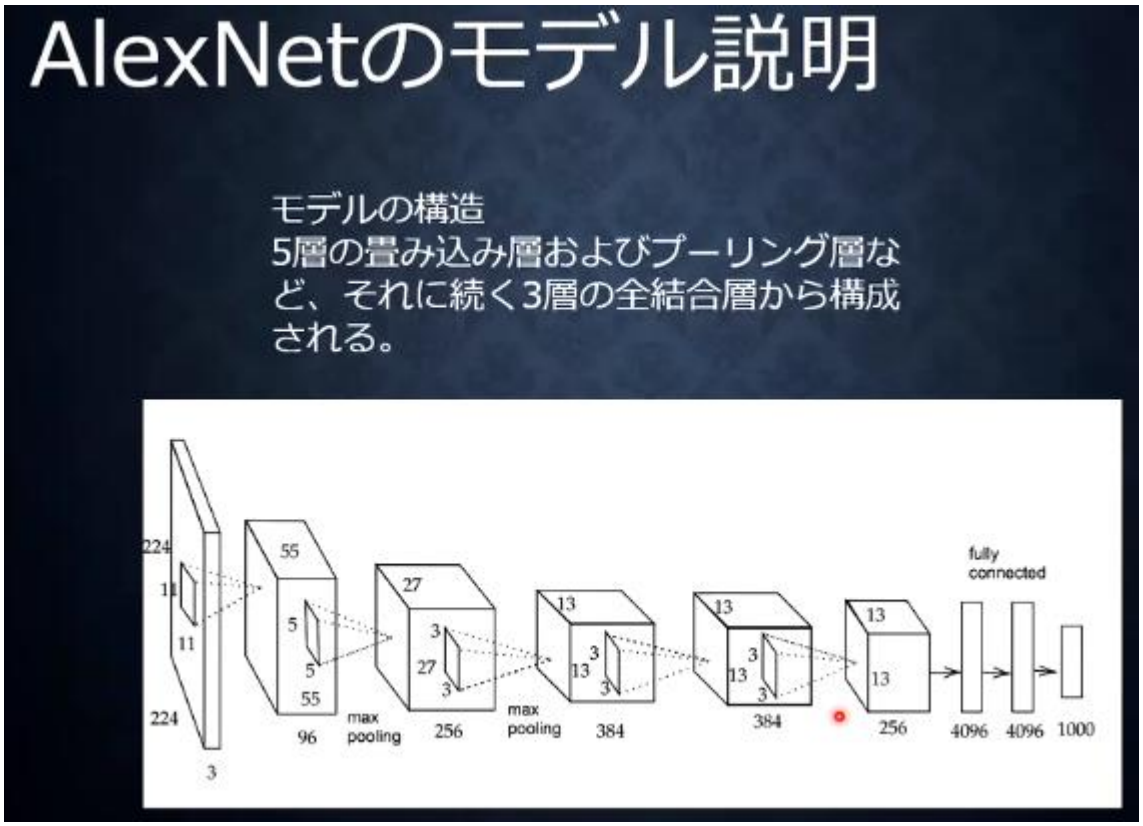
公式

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

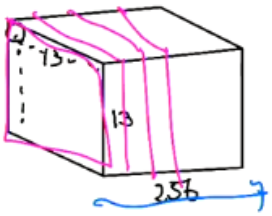
$$O_W = \frac{\text{画像の幅} + 2\times\text{パディング} - \text{フィルター幅}}{\text{ストライド}}$$

```
# 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)
```

４．８　Alexnet



1）全結合層への変換



Fratten → [0 0 0 .... 0 0 0]
43,264

Global MaxPooling → 1番大 [0 0 0 ... 0 0]
256

Global Avg Pooling → 平均 [0 0 ... 0 0]
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)

①．ReLU

②．LRN(Local Response Normalization)

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

③．Overlapping Pooling

Pooling 層をオーバラップさせる。

④．DropOut

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。

②．２層め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)

①．Inseption モジュール

複数の畳み込み層や Pooling 層から構成される Inseption 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。

②．Global Average Pooling

③．Auxiliary Loss

④．Inseption-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。

③．max pooling の後の畳み込み層の出力チャネル数を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。

①．Residual モジュール

上記ショートカットの名称

②. Batch 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