深層学習の復習

深層学習の流れ

1. 入力層に値を入力
2. 重み、バイアス、活性化関数で計算しながら値が伝わる。
3. 出力層から値が伝わる。
4. 出力層から出た値と正解値から、誤差関数を使って誤差を求める。
5. 誤差を小さくするために重みやバイアスを更新する。
6. 1～5 の操作を繰り返すことにより、出力値を正解値に近づけていく。

誤差逆伝播の復習

計算結果(=誤差)から微分を逆算することで、不要な再帰的計算を避けて微分を算出できる。

$E(y) = \frac{1}{2}\sum_{j=1}^{J}(y_j - d_j) = \frac{1}{2}\|(y-d\|^2$　　誤差関数 ＝ 二乗誤差関数

$y = u^{(L)}$　　出力層の活性化関数 ＝ 恒等写像

$u^{(l)} = w^{(l)}z^{(l-1)} + b^{(l)}$　　総入力の計算

$$\frac{\partial E}{\partial w_{ji}^{(2)}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial w_{ji}^{(2)}}$$

$$\frac{\partial E(y)}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}\|y-d\|^2 = y - d$$

$$\frac{\partial y(u)}{\partial u} = \frac{\partial u}{\partial u} = 1$$

$$\frac{\partial u(w)}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(w^{(l)}z^{(l-1)} + b^{(l)}\right) = \frac{\partial}{\partial w_{ji}}\left(\begin{bmatrix} w_{11}z_1 + \cdots + w_{1i}z_i \cdots + w_{1I}z_I \\ \vdots \\ w_{j1}z_1 + \cdots + w_{ji}z_i \cdots + w_{jI}z_I \\ \vdots \\ w_{J1}z_1 + \cdots + w_{Ji}z_i \cdots + w_{JI}z_{Ii} \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \\ \vdots \\ b_J \end{bmatrix}\right) = \begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix}$$

Section1：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。
活性化関数であるシグモイド関数は、大きな値では出力値の変化が微小なため、勾配消失問題を引き起こすことがあった。

シグモイド関数

0〜1 の間を緩やかに変換する関数で、ステップ関数では ON/OFF しかなかった状態に対して、信号の強弱を伝えられるようになり、予測ニューラルネットワーク普及のきっかけとなった。

$$f(u) = \frac{1}{1 + e^{-u}}$$

下記のようにシグモイド関数(0〜1)の積算が繰り返し行われることで、値が小さくなる。

$$\frac{\partial E}{\partial y}\frac{\partial y}{\partial u^{(3)}}\frac{\partial u^{(3)}}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial u^{(2)}} \qquad \frac{\partial z^{(2)}}{\partial u^{(2)}} = \big(1 - sigmoid(x)\big) \cdot sigmoid(x)$$

$$\frac{\partial E}{\partial y}\frac{\partial y}{\partial u^{(3)}}\frac{\partial u^{(3)}}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial u^{(2)}}\frac{\partial u^{(2)}}{\partial z^{(1)}}\frac{\partial z^{(1)}}{\partial u^{(1)}} \qquad \frac{\partial z^{(1)}}{\partial u^{(1)}} = \big(1 - sigmoid(x)\big) \cdot sigmoid(x) \quad \frac{\partial z^{(2)}}{\partial u^{(2)}} = \big(1 - sigmoid(x)\big) \cdot sigmoid(x)$$

勾配消失の解決方法

・活性化関数の選択
・重みの初期値設定
・バッチ正規化

活性化関数

シグモイド関数ではなく、ReLU 関数を選択する。

ReLU 関数

最も使用されている活性化関数。勾配消失問題の回避とスパース化に貢献することで良い成果をもたらしている。

$$f(x) = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$$

$$\frac{\partial z^{(1)}}{\partial u^{(1)}} = 0 \; or \; x \quad \frac{\partial z^{(2)}}{\partial u^{(2)}} = 0 \; or \; x$$

初期値の設定方法

重みの初期値の設定方法としては、Xavier と He

Xavier

重みの要素を前の層のノード数を n ことした場合に、平均=0, 標準偏差=$\frac{1}{\sqrt{n}}$である正規分布から作成する。

シグモイド関数、ReLU 関数、双曲線正接関数などを活性化関数として用いた時に効果を発揮する。

初期値の設定の実装例

```
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)
```

He

重みの要素を前の層のノード数を n 個とした場合に、平均=0, 標準偏差=$\frac{1}{\sqrt{\frac{n}{2}}}$である正規分布から作成する。

ReLU 関数などを活性化関数として用いた時に効果を発揮する。

初期値の設定の実装例

```
network["W1"] = np.random.randn(input_layer_size, hidden_layer_size) / np.sqrt(input_layer_size) * np.sqrt(2)
network["W2"] = np.random.randn(hidden_layer_size, output_layer_size) / np.sqrt(hidden_layer_size) * np.sqrt(2)
```

バッチ正規化

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。

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。

$u^{(l)} = w^{(l)}z^{(l-1)} + b^{(l)}$または zがバッチ正規化層への入力値である。

学習の安定化や速度アップできる。
また、過学習を抑えることができる。

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

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

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

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

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$\left(0\right.$ を中心とするセンタリング$\left.\right)$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出力)

<確認テスト 1>

Q. 連鎖律の原理を使い、$\frac{dz}{dx}$を求めよ。

$z = t^2$

$t = x + y$

A. $2(x + y)$

$\frac{dz}{dt} = 2t$、$\frac{dt}{dx} = 1$、$\frac{dz}{dx} = \frac{dz}{dt}\frac{dt}{dx}$より、$\frac{dz}{dx} = 2t \times 1 = 2t = 2(x + y)$

<確認テスト 2>

Q. シグモイド関数を微分した時、入力値が 0 の時に最大値をとる。その値として正しいものを選択肢から選べ。

(1) 0.15

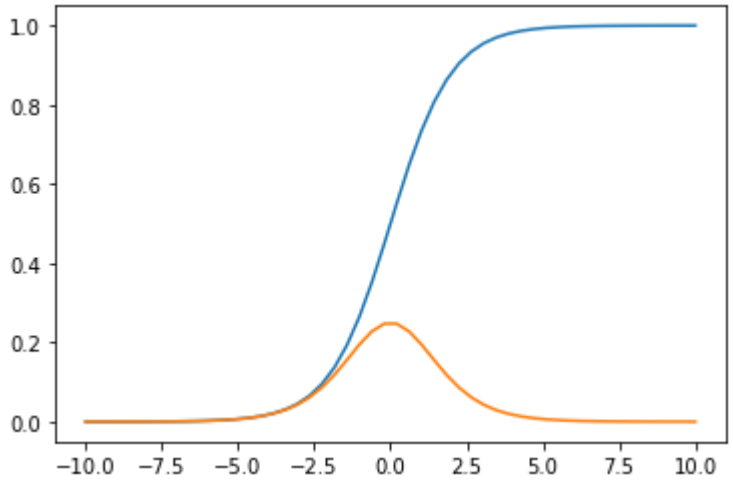
(2) 0.25

(3) 0.35

(4) 0.45

A. (2) 0.25

シグモイド関数を微分すると、$f'(u) = (1 - \frac{1}{1+e^{-u}})\frac{1}{1+e^{-u}}$であり、これをプロットすると最大値が 0.25 であることがわかる。



青線　　：シグモイド関数

オレンジ線　　：シグモイド関数の微分値

<確認テスト 3>

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

A. 重みを 0 で初期化すると正しい学習が行えない。全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。

<確認テスト 4>

Q. 一般的に考えられるバッチ正規化の効果を 2 点あげよ。

A. 以下の 2 点がある。

1. バッチ正則化を行うことで、ニューラルネットワークの学習中に、中間層の重みの更新が安定化される。その結果として学習スピードがアップする。
2. 過学習を抑えることができる。バッチ正則化はミニバッチの単位でデータの分布を正規化するため、学習データの極端なばらつきが抑えられる。
   この状態でニューラルネットワークの学習ができるため、過学習が起きにくい調整が行われる。

<練習問題>

Q. 深層学習では一般に要するデータが多く、メモリなどの都合ですべてまとめてバッチで計算することはできない。
そのためデータを少数のまとまりであるミニバッチにして計算を行う。
以下は特徴データ data_x、ラベルデータ data_t に対してミニバッチ学習を行うプログラムである。
(き)にあてはまるのはどれか。

```
def train(data_x, data_t, n_epoch, batch_size):
 “””
 data_x: training data (features)
 data_t: training data ( labels)
 n_epoch: number of epochs
 batch_size: mini batch size
 “””
 n = len(data_x)
```

```
for epoch in range(n_epoch):
    shuffle_idx = np.random.permutation(N)
    for i in range(0, N, batch_size):
        i_end = i + batch_size
        batch_x, batch_t = (き)
        _update(batch_x, batch_t)
```

(1) data_x[i:i_end], data_t[i:i_end]
(2) data_x[i_end:i], data_t[i:i_end]
(3) data_x[i:], data_t[i:]
(4) data_x[:i_end], data_t[:i_end]

A. (1) data_x[i:i_end], data_t[i:i_end]
batch_size 分だけデータを取り出す処理であり、i_end = i + batch_size であるため、
取り出すデータの先頭のインデックス=i, 取り出すデータの最後のインデックス+1= i_end であるため、(1)が正解となる。
(2)だと逆順での取得となっている、(3)は batch_size 分だけではなくインデックス=i から最後までのデータの取り出し、(4)だと先頭からインデックス=i_end の一つ前までのデータの取り出しのため、batch_size 分のデータの取り出しにならない。

<実装演習>
2_1_network_modified.ipynb
2_2_1_vanishing_gradient.ipynb
2_2_2_vanishing_gradient_modified.ipynb
2_3_batch_normalization.ipynb

Section2：学習率最適化手法
学習率とは勾配降下法の下記式に登場するεのこと。
$w^{(t+1)} = w^{(t)} + \varepsilon \nabla E$
・学習率が大きすぎる場合
最適値にいつまでもたどり着かず発散する。

・学習率が小さすぎる場合
　収束するまでに時間がかかったり、大域局所最適値に収束しづらくなる。

学習率の設定方法
　初期の学習率は大きく設定し、徐々に学習率を小さくしていく。
　パラメータ毎に学習率を可変させる。
　上記の方法で学習率の最適化を行うために、モメンタム、AdaGrad、RMSProp、Adam などの手法を利用する。

モメンタム
　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惰性の積を加算する方法。

　<数式>

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

　$w^{(t+1)} = w^t + V_t$　　②

　μ：惰性

　<Python コード>
　① self.h[key] = self.momentum * self.v[key] – self.learning_rate * grad[key]
　② params[key] += self.v[key]

　メリット
　・局所的最適解にはならず、大域的最適解となる。
　・谷間についてから最も低い位置(最適値)にいくまでの時間が早い。

AdaGrad
　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。

<数式>

$h_0 = \theta$ ①

$h_t = h_{t-1} + (\nabla E)^2$ ②

$w^{(t+1)} = w^t - \varepsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$ ③

<Python コード>

③ self.h[key] = np.zeros_like(val)

④ self.h[key] += grad[key] * grad[key]

⑤ params[key] -= self.learning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e–7)

メリット

・勾配の緩やかな斜面に対して、最適値に近づける。

課題

・学習率が徐々に小さくなるため、鞍点問題を引き起こすことがある。

RMSProp

AdaGrad を改良したもの。

<数式>

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

$w^{(t+1)} = w^t - \varepsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$ ②

α：前回までの勾配情報をどのくらい参考にするか(0～1 の範囲の値)。

<Python コード>

⑥ self.h[key] *= self.decay_rate
self.h[key] += (1 – self.decay_rate) * grad[key] * grad[key]
⑦ params[key] -= self.learning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e–7)

メリット
・局所的最適解にはならず、大域的最適解となる。
・ハイパーパラメータの調整が必要な場合が少ない。

Adam
モメンタムと RMSProp のメリットを合わせた最適化アルゴリズム。
・モメンタムの過去の勾配の指数関数的減衰平均
・RMSProp の過去の勾配の 2 乗の指数関数的減衰平均

<確認テスト>
Q. モメンタム・AdaGrad・RMSProp の特徴をそれぞれ簡単に説明せよ。

A. それぞれ以下のメリットがある。
<モメンタムのメリット>
・局所的最適解にはならず、大域的最適解となる。
・谷間についてから最も低い位置(最適値)にいくまでの時間が早い。

<AdaGrad のメリット>
・勾配の緩やかな斜面に対して、最適値に近づける。

<RMSProp のメリット>
・局所的最適解にはならず、大域的最適解となる。
・ハイパーパラメータの調整が必要な場合が少ない。

<実装演習>
2_4_optimizer.ipynb

2_4_optimizer_after.ipynb

Section3：過学習

過学習

テスト誤差と訓練誤差とで学習曲線が乖離（=訓練データに対しては誤差が小さいが、テストデータに対しては誤差が大きい)し、正しく予測できていない状態のこと。

パラメータ数が多い、パラメータの値が適切でない、ノード数が多いなどのネットワーク自由度が高いことが原因で発生する。

正則化やドロップアウトという手法を用いることで過学習を抑制する。

Weight decay(荷重減衰)

重みが大きい値は学習において重要な値であることを示すが、値が大きいと過学習が発生（=特定のパラメータに過剰反応している状態）することがある。

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はばらつきを持たせる必要があるため、誤差に対して、正則化項を加算することで、重みの大きさを抑制するという対応を行う。

正則化

パラメータの値、ノード数、層数などのネットワーク自由度を制約することを正則化という。

過学習を抑制するために利用する。

L1 正則化、L2 正則化

下記の計算式において、p=1 の場合を L1 正則化、p=2 の場合を L2 正則化と呼ぶ。

$E_n(w) + \frac{1}{p}\lambda\|x\|_p$　　式①：誤差関数に、p ノルムを加える>

$\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$　　式②：p ノルムの計算式

それぞれ python のコードにすると以下となる。

式①：np.sum(np.abs(network.params['W' + str(idx)]))

式②：np.sum(np.abs(network.params['W' + str(idx)]))

ノルムとはベクトル空間での距離のこと。

P1 ノルムはマンハッタン距離(=2 点間を水平と垂直で移動する距離)であり、L1 正則化を施した回帰は Lasso(ラッソ)回帰と呼ぶ。

P2 ノルムはユークリッド距離(=2 点間の直線距離)であり、L2 正則化を施した回帰は Ridge (リッジ)回帰と呼ぶ。

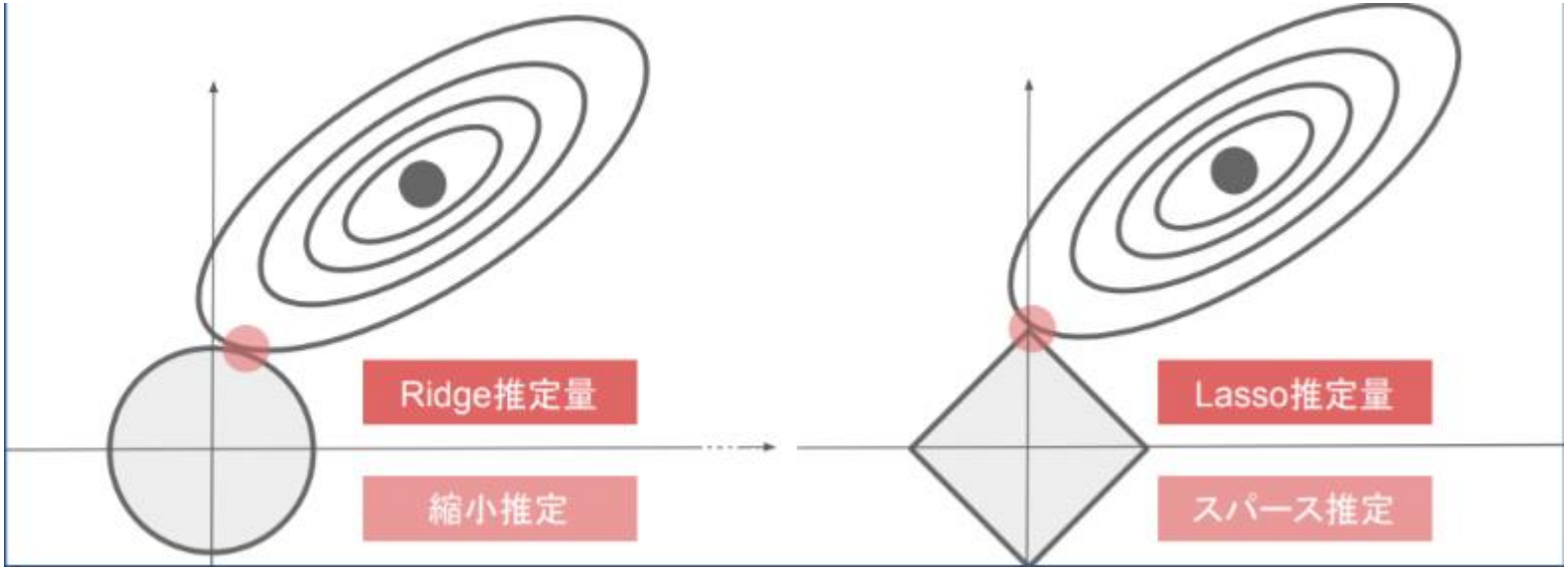
ドロップアウト

ランダムにノードを削除して学習させることをドロップアウトという。

データ量を変化させずに、異なるモデルを学習させている状態にできる。

<確認テスト 1>

Q. 下図について、L1 正則化を表しているグラフはどちらか答えよ。



A. 右の Lasso 推定量。

Lasso はスパース推定を実現する手法の一つ。

L1 正則化の右図は誤差が最小となっているのが横軸の値が 0 の時のため、横軸方向の重みが 0 になることを表している。

<練習問題 1>

Q. 深層学習において、過学習の抑制・汎化性能の向上のために正則化が用いられる。そのひとつに、L2 ノルム正則化(Ridge, Weigh Decay)がある。

以下は L2 正則化を適用した場合に、パラメータの更新を行うプログラムである。あるパラメータ param と正則化がないときにそのパラメータに伝播される誤差の勾配 grad が与えられたとする。
最終的な勾配を計算する(え)に当てはまるのはどれか。ただし rate は L2 正則化の係数を表すとする。

```
def ridge(param, grad, rate):
  “””
  param: target parameter
  grad: gradients to param
  rate: ridge coefficient
  “””
  grad += rate * (え)
```

(1) np.sum(param**2)
(2) np.sum(param)
(3) param**2
(4) param

A. (4) param
L2 ノルムは$\|x\|^2$なのでその勾配が誤差の勾配に加えられるので、x^2を微分した$2x$が加えられると考えられ、係数 2 は正則化の係数に吸収されても変わらないため、xのみと考えて(え)には param が正解となる。

<練習問題 2>
Q. 以下は L1 ノルム正則化(Lasso)を適用した場合に、パラメータの更新を行うプログラムである。あるパラメータ param と正則化がないときにそのパラメータに伝播される誤差の勾配 grad が与えられたとする。
最終的な勾配を計算する(お)に当てはまるのはどれか。ただし rate は L1 正則化の係数を表すとする。

```
def lasso(param, grad, rate):
  “””
  param: target parameter
```

```
grad: gradients to param
rate: ridge coefficient
“””
grad += rate * (お)
```

(1) np.maximum(param, 0)
(2) np.minimum(param, 0)
(3) np.sign(param)
(4) np.abs(param)

A. (3) np.sign(param)
L1 ノルムは$\|x\|$なのでその勾配が誤差の勾配に加えられるので、xの正負により± 1えられると考えられる。
numpy の sign()は、引数の値が負の場合は-1、正の場合は 1、0 の場合は 0 を返す関数であるため、(3)の np.sign(param)が正解となる。

<練習問題 2>
Q. 画像認識などにおいて、精度向上や汎化性能の向上のためにデータ拡張が行われることが多い。
データ拡張には、画像を回転・反転させるなど様々な種類がある。以下は画像をランダムに切り取る処理を行うプログラムである。
これは画像中の物体の位置を移動させるなどの意味がある。
(か)に当てはまるものはどれか。

```
def random_crop(image, crop_size):
  “””
  image: (height, width, channel)
  crop_size: (crop_height, crop_width)
  height >= crop_height, width >= crop_width
  “””
  h, w, _ = image.shape
  crop_h, crop_w = crop_size
```

```
# 切り取る位置をランダムに決める
top = np.random.randint(0, h – crop_h)
left = np.random.randint(0, w – crop_w)
bottom = top + crop_h
right = left + crop_w

image = (か)
return image
```

(1) image[:, top:bottom, left:right]
(2) image[:, bottom:top, right:left]
(3) image[bottom:top, right:left, :]
(4) image[top:bottom, left:right, :]

A. (4) image[top:bottom, left:right, :]
画像をランダムに切り取って返す処理であり、image[]は高さ、幅、チャンネルの順で値が設定されているため、縦幅が 1 つめ、横幅が 2 つ目に設定されている必要があるため、(4)が該当する。

<実装演習>
2_5_overfiting.ipynb

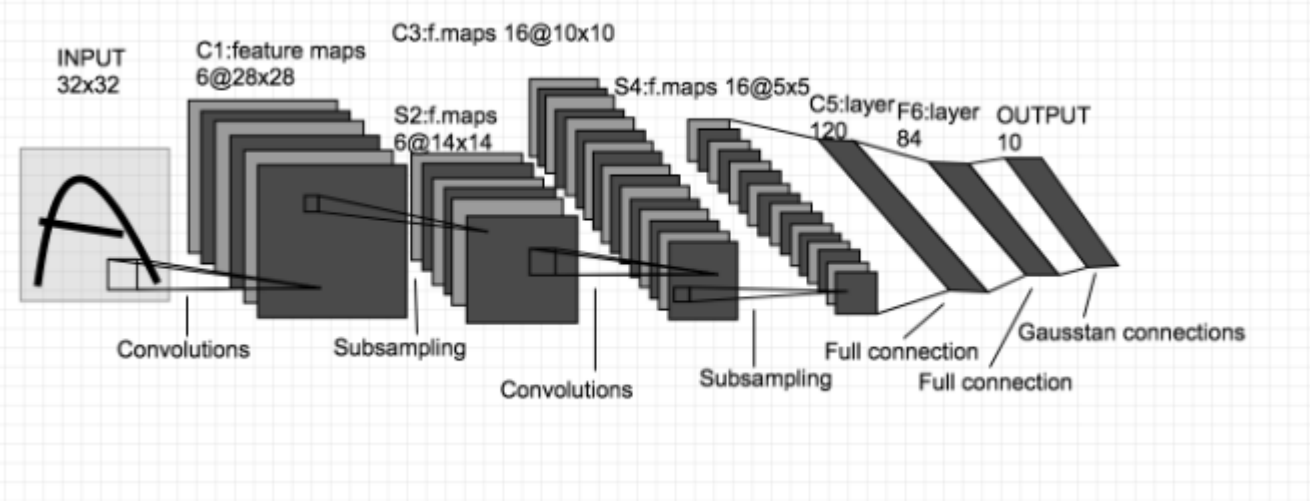
Section4：畳み込みニューラルネットワークの概念
畳み込みニューラルネットワークは、画像の識別や画像の処理のために使われるニューラルネットワーク。
音声データ、画像データ、動画データなどのように次元の繋がりがあれば使用できる。

畳み込みニューラルネットワークの構造
構造の例としては下記であり、入力層と出力層の間に畳み込み層/プーリング層/全結合層がある構造となっている。
1. 入力層(入力画像)

2. 畳み込み層
3. 畳み込み層
4. プーリング層
5. 畳み込み層
6. 畳み込み層
7. プーリング層
8. 全結合層
9. 出力層(出力画像)

代表的な畳み込みニューラルネットワーク

下記の構造図となる LeNet というものが畳み込みニューラルネットワークの代表的なものとしてある。



32×32 の画像の入力を受け取って、10 種類に分類できる構造となっている。

(32×32=1024 のデータを 10 個のデータに置き換えているということ。)

Convlutions は畳み込み演算のことで、下記の処理により最終的に 10 個のデータへの分類が行われる。

1. [32, 32, 1]→[28, 28, 6]のデータに変換されており、28×28×6=4704 個のデータに変換。
2. [28, 28, 6]→[14, 14, 6]のデータに変換されており、14×14×6=1176 個のデータに変換。
3. [14, 14, 6]→[10, 10, 16]のデータに変換されており、10×10×16=1600 個のデータに変換。
4. [10, 10, 16]→[5, 5, 16]のデータに変換されており、5×5×16=400 個のデータに変換。

5. [5, 5, 16]→120 個のデータに変換。
6. 120 個のデータを 84 個のデータ、84 個のデータを 10 個のデータに変換。

畳み込み層

畳み込み層では以下の計算が行われる。

入力値 × フィルター(全結合でいう重み) → 出力値 + バイアス → 活性化関数 → 出力値

※フィルターは入力値で参照する一部のデータ領域を示し、フィルターをずらして入力値を参照すること。

例）入力値=4×4 のデータ、フィルター=3×3 のデータ → 出力値= 2×2 のデータ

畳み込み層では、画像の場合は縦、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。

→3 次元の空間情報も学習できるような層が畳み込み層ということ。

フィルター

全結合でいう重みであり、フィルターの各位置での重み。

フィルターは複数存在可能。

例)各位置に 1～8 の重みが設定されている 3×3 のフィルター

3	1	2
8	7	5
5	4	1

バイアス

入力値 × フィルターで算出された値に加算する値のこと。

バイアス値は 1 つであり、フィルターが複数ある場合でも全てに同じバイアス値を適用する。

パディング

入力値 × フィルターの計算を行うと、元のデータサイズよりも少なくなる。

これを解消するために、入力値の周りに固定の値が設定された状態で入力値 × フィルターの計算を行い、元データサイズから変更が発生しないようにすること。

例）入力値が 4×4(赤字部分)に 0 でパディングした場合 。

パディングして 6×6 のデータに 3×3 のフィルターを適用した場合、4×4 のデータとなる。

0	0	0	0	0	0
0	3	4	4	2	0
0	0	8	9	2	0
0	0	4	3	1	0
0	3	3	5	7	0
0	0	0	0	0	0

ストライド

フィルターを移動させる移動量のこと。

ストライドのサイズが大きいほど、入力値×フィルターのデータは元データサイズよりも小さくなる。

チャンネル

適用するフィルターの数のこと。

プーリング層

畳み込み層のように少しずつずれながら計算するが、畳み込み層のように重みによる計算ではなく、以下のどちらかの計算を行った結果を出力する。

マックスプーリング(MaxPooling)　　：畳み込み層の出力を入力とし、そのうちの最大値を出力する。

アベレージプーリング(AvgPooling)　　：畳み込み層の出力を入力とし、それらの入力の平均値を出力する。

全結合層

畳み込み層やプーリング層では、次元の繋がりが保たれた特徴量が算出され、この特徴量から人間が欲しい結果を求める部分。

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ため、RGB の各チャンネル間の関連性が学習に反映されない。

<確認テスト 1>

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. 7×7

パティング 1 のため、8×8 の入力画像にストライド 1 で 2×2 のフィルターを適用するということになる。

縦方向、横方向どちらも 7 個となるため、7×7 となる。

出力サイズについては下記の式から、(6 + 2×1 - 2) / 1 + 1 = 7 と計算することも可能。

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

OH：出力サイズの高さ、OW：出力サイズの幅

H：入力サイズの高さ、W：入力サイズの幅

FH：フィルターの高さ、FW：フィルターの幅

P：パディング、S：ストライド

<実装演習>

2_6_simple_convolution_network.ipynb

2_6_simple_convolution_network_after.ipynb

Section5：最新の CNN

AlexNet

5 層の畳み込み層及びプーリング層、それに続く 3 層の結合層からなるモデル構造。

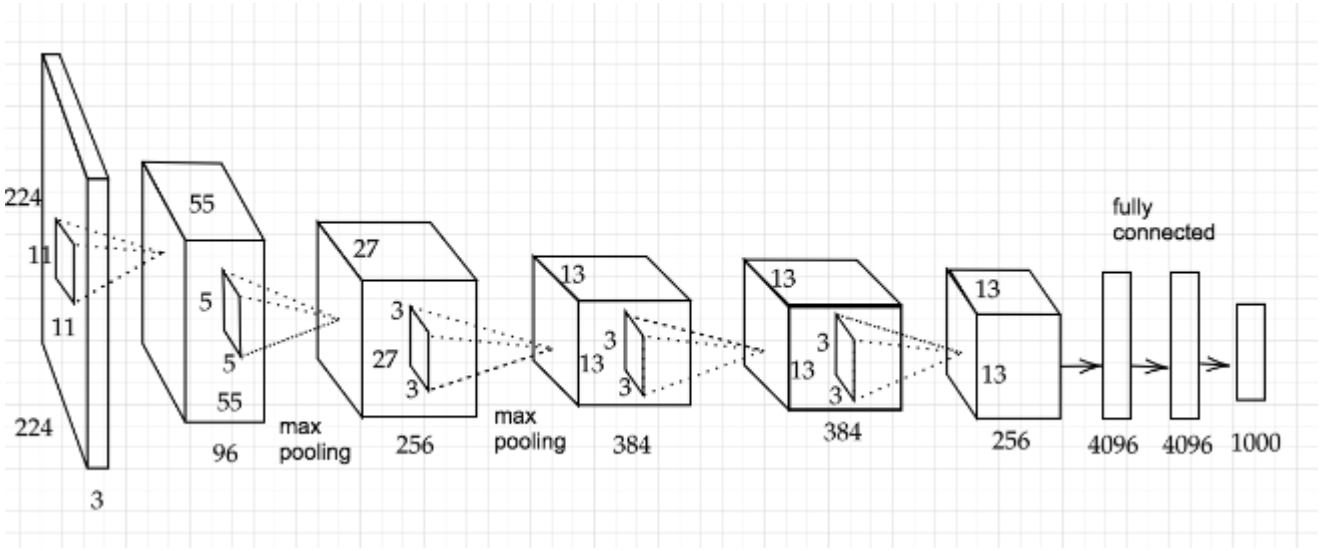
1. [224, 224, 3]の画像を 11×11 のフィルターで畳み込み演算して、[55, 55, 96]の画像に変換する。
2. [55, 55, 96]の画像を 5×5 のフィルターで畳み込み演算して、[27, 27, 256]の画像に変換する。
3. [27, 27, 256]の画像を 3×3 のフィルターで畳み込み演算して、[13, 13, 384]の画像に変換する。
4. [13, 13, 384]の画像を 3×3 のフィルターで畳み込み演算して、[13, 13, 384]の画像に変換する。
5. [13, 13, 384]の画像を 3×3 のフィルターで畳み込み演算して、[13, 13, 256]の画像に変換する。
6. [13, 13, 256]の画像を Flatten で全結合する。

   Flatten：横一列にデータを並べる手法。

   Global Max Pooling：全結合するための手法の一つで、各チャンネルの最大値を利用する。

   Global Avg Pooling：全結合するための手法の一つで、各チャンネルの平均値を利用する。

7. 過学習の抑制のため、ドロップアウトでサイズ 4096 からサイズ 1000 にする。



LeNet というものが先行してあり、それとは以下の点が異なる。

・活性化関数に ReLU を用いる。

・LRN(Local Response Normalization)という局所的正規化を行う層を用いる。

・ドロップアウトを使用する。

<実装演習>

2_7_double_comvolution_network.ipynb

2_7_double_comvolution_network_after.ipynb

2_8_deep_convolution_net.ipynb