バッチ正規化

深層学習において学習効率を向上させる強力な仕組みであるバッチ正規化について考える。

1. バッチ正規化

バッチ正規化（batch normalization）の目的は統計的分布の時間的変化による影響を標準化によって吸収させることである。入力層に渡される統計的な分布が時間とともに変化していくと、ニューラルネットワークの学習が困難になることは古くから知られていた。このような現象は共変量シフト（covariate shift）と呼ばれる。共変量とは統計学の用語であり、データの属性のうち、予測対象でないものを指す。入力の統計分布が変化すると学習器はそれに対応してパラメータを変化させなくてはならず、それがコストとなって学習が非効率化する。予測器の目標は真のメカニズムを表す分布 $q(\boldsymbol{y}|\boldsymbol{x})$ に自らの入出力関係 $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ を近づけることである。$q(\boldsymbol{y}|\boldsymbol{x})$ は $\boldsymbol{x}$ の各値についての $\boldsymbol{y}$ の分布であり、入力の分布 $q(\boldsymbol{x})$ とは異なるものであるが、$q(\boldsymbol{x})$ の変化は $q(\boldsymbol{y}|\boldsymbol{x})$ の近似も難しくするということである。

統計的分布の変化が問題となるのは入力層に限らない。学習が進むにつれ、ユニット間の結合の重みが変わるため、隠れユニットが受け取る活性の統計分布は変化していく。この現象を内部共変量シフト（internal covariate shift）と呼ぶ。出力に近い側の層における結合はそれより入力側の活性の統計分布の変化に合わせて更新される必要があり、これが学習の非効率化を招く。そこでバッチ正規化（batch normalization）では内部共変量シフトに対する対策として、隠れ層の間に標準化を挟む。標準化はベクトルにおける正規化に類似するため、この名称がつけられている。また、バッチとは確率的勾配降下法におけるミニバッチに相当し、小さな集合を表す。

標準化は活性の成分ごとにおこなう。i 番目のサンプルにおける第 l 層の j 番目のユニットの活性を $a_j^{(l,i)}$ で表す。すなわち、第 l 層のユニット数を m として

$$a_j^{(l,i)} \quad \begin{array}{l} i：\text{サンプル番号} \\ l：\text{層番号} \\ j：\text{ユニット番号}\ \ (j = 1, \cdots, m) \end{array}$$

である。複数のサンプルから得られた活性をまとめて得られるバッチを $\mathcal{B}$ で表す。$\tilde{n}$ をバッチサイズとすると、バッチの要素は $a_j^{(l,1)}, \cdots, a_j^{(l,\tilde{n})}$ で表せる。バッチにおける平均値 $\mu_{j\mathcal{B}}^{(l)}$ と分散 $v_{j\mathcal{B}}^{(l)}$ は以下のように計算される。

$$\mu_{j\mathcal{B}}^{(l)} = \frac{1}{\tilde{n}} \sum_{k=1}^{\tilde{n}} a_j^{(l,k)}, \quad v_{j\mathcal{B}}^{(l)} = \frac{1}{\tilde{n}} \sum_{k=1}^{\tilde{n}} \left(a_j^{(l,k)} - \mu_{j\mathcal{B}}^{(l)} \right)^2 \tag{1}$$

標準化された $\hat{a}_j^{(l,i)}$ は以下のように求まる。ただし ϵ は数値計算の安定性のために入れる小さな値であり、あまりに 0 に近い値での割り算が行われないようにするためのものである。

$$\hat{a}_j^{(l,i)} = \frac{a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)}}{\sqrt{v_{j\mathcal{B}}^{(l)} + \epsilon}} = \frac{a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)}}{s_{j\mathcal{B}}^{(l)}} \tag{2}$$

ここで

$$s_{j\mathcal{B}}^{(l)} = \sqrt{v_{j\mathcal{B}}^{(l)} + \epsilon} \tag{3}$$

である。

データの集合の各サンプルを標準化すると、変換後の集合の平均値は 0、標準偏差は 1 になる。結果としてどの隠れユニットも平均が 0、標準偏差が 1 の活性を受け取ることになる。しかしそのような分布が隠れユニットの活性化関数に対して適切とは限らない。たとえばシグモイド関数は区間 $[-1, 1]$ ではほとんど直線的である。もっと大きな入力を与えなければ、その非線形性が発揮されない。そこでパ

ラメータ $\beta_j^{(l)}$ と $\gamma_j^{(l)}$ を導入し、$\hat{a}_j^{(l,i)}$ をさらに以下のように変換する。このような変換はアファイン変換と呼ばれる。

$$y_j^{(l,i)} = \gamma_j^{(l)} \hat{a}_j^{(l,i)} + \beta_j^{(l)} = \frac{\gamma_j^{(l)}(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)})}{s_{j\mathcal{B}}^{(l)}} + \beta_j^{(l)} \tag{4}$$

式 (4) の操作をバッチ正規化変換と呼ぶ。

標準化は変換後のデータ集合を平均が 0 で標準偏差が 1 の分布に従わせることであるが、バッチ正規化変換は平均が $\beta_j^{(l)}$ で標準偏差が $\gamma_j^{(l)}$ である分布に従わせる変換であると言える。

バッチ正規化変換の出力 $y_j^{(l,i)}$ が活性化関数 $\psi^{(l)}$ の入力となり第 l 層の出力

$$z_j^{(l,i)} = \psi^{(l)}(y_j^{(l,i)})$$

が得られる。

パラメータ $\beta_j^{(l)}$ と $\gamma_j^{(l)}$ は各層の活性の成分ごと、すなわちユニットごとに用意される。$\beta_j^{(l)}$ と $\gamma_j^{(l)}$ はパラメータ更新によって漸進的に変化していくが、それを除けば活性は固定的な分布に従うことになる。つまり活性の分布が急激に変化しても、それは標準化によって吸収される。しかしその変化が継続的であり、またパフォーマンスを向上させるものであれば、$\beta_j^{(l)}$ と $\gamma_j^{(l)}$ が変化することでゆっくりと対応できる。これは $a_j^{(l)}$ の統計分布の変化への対応を重み行列ではなく $\beta_j^{(l)}$ と $\gamma_j^{(l)}$ に担当させるという見方もできる。また、$\mu_{j\mathcal{B}}^{(l)}$ と $v_{j\mathcal{B}}^{(l)}$ はバッチ $\mathcal{B}$ ごとに計算されるが、$\beta_j^{(l)}$ と $\gamma_j^{(l)}$ はバッチをまたがって共通である。このためバッチ間で分布のばらつきがあるとき、それを吸収する働きがある。

式 (4) から分かるように、もし $\mu_{j\mathcal{B}}^{(l)} = \beta_j^{(l)}$ かつ $v_{j\mathcal{B}}^{(l)} = \gamma^2$ で $\epsilon = 0$ の場合

$$y_j^{(l,i)} = a_j^{(l,i)}$$

となる。これは変換が行われないことを意味する。つまりバッチ正規化変換には「変換を全く行わない」という選択肢も含まれるため、モデルの表現力を低下させない。すなわちバッチ正規化変換を挟んだことで一部の入出力関係が表現できなくなることはない。逆にいえばもしバッチ正規化変換の一部として $\beta_j^{(l)}$ と $\gamma_j^{(l)}$ によるアファイン変換を入れない場合、モデルの表現力が低下してしまう可能性がある。

バッチ正規化変換を活性化層すなわち隠れ層への入力に対してではなく、隠れ層の出力に対して行うという方式も考えられるが、活性に対して行うほうが良いとされている。これは隠れ層の出力（活性化関数の値）は正規分布と大きく異なる分布に従うため、平均と標準偏差を使って標準化することが適切でないと考えられるためである。これに対し、活性は多数の活性化関数の値が足し合わされることで得られているため、その分布が正規分布に近くなり、標準化が有効と考えられる。

なお、式 (4) は訓練時に行られる変換であるが、予測時にはバッチではなく全データを使って平均と標準偏差を求め、同様の変換を行う。

2. バッチ正規化変換における誤差逆伝播法

バッチ正規化変換を含むニューラルネットワークにおいて、パラメータ $\beta_j^{(l)}$, $\gamma_j^{(l)}$ および重みパラメータを更新するための誤差逆伝播法について考える。

ここまでの関係を再記して整理すると、順伝播は

$$\mu_{j\mathcal{B}}^{(l)} = \frac{1}{\tilde{n}} \sum_{\lambda=1}^{\tilde{n}} a_j^{(l,\lambda)} \tag{5}$$

$$v_{j\mathcal{B}}^{(l)} = \frac{1}{\tilde{n}} \sum_{\lambda=1}^{\tilde{n}} \left(a_j^{(l,\lambda)} - \mu_{j\mathcal{B}}^{(l)} \right)^2 \tag{6}$$

$$s_{j\mathcal{B}}^{(l)} = \sqrt{v_{j\mathcal{B}}^{(l)} + \epsilon} \tag{7}$$

$$\hat{a}_j^{(l,k)} = \frac{a_j^{(l,k)} - \mu_{j\mathcal{B}}^{(l)}}{s_{j\mathcal{B}}^{(l)}} \tag{8}$$

$$y_j^{(l,k)} = \gamma_j^{(l)} \hat{a}_j^{(l,k)} + \beta_j^{(l)} \tag{9}$$

となる。逆伝播で勾配の計算に必要となる微分を計算するために、式 (5)-(9) を $a_j^{(l,i)}$ で微分する。

$$\frac{\partial \mu_{j\mathcal{B}}^{(l)}}{\partial a_j^{(l,i)}} = \frac{1}{\tilde{n}} \tag{10}$$

$$\begin{aligned}
\frac{\partial v_{j\mathcal{B}}^{(l)}}{\partial a_j^{(l,i)}} &= \frac{2}{\tilde{n}} \sum_{\lambda=1}^{\tilde{n}} \left(\frac{\partial a_j^{(l,\lambda)}}{\partial a_j^{(l,i)}} - \frac{\partial \mu_{j\mathcal{B}}^{(l)}}{\partial a_j^{(l,i)}} \right) \left(a_j^{(l,\lambda)} - \mu_{j\mathcal{B}}^{(l)} \right) \\
&= \frac{2}{\tilde{n}} \sum_{\lambda=1}^{\tilde{n}} \left(\delta_{i\lambda} - \frac{1}{\tilde{n}} \right) \left(a_j^{(l,\lambda)} - \mu_{j\mathcal{B}}^{(l)} \right) \\
&= \frac{2}{\tilde{n}} \left(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)} \right) - \frac{2}{\tilde{n}^2} \sum_{\lambda=1}^{\tilde{n}} \left(a_j^{(l,\lambda)} - \mu_{j\mathcal{B}}^{(l)} \right) \\
&= \frac{2}{\tilde{n}} \left(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)} \right)
\end{aligned} \tag{11}$$

ここで、$\mu_{j\mathcal{B}}^{(l)}$ の定義より

$$\sum_{\lambda=1}^{\tilde{n}} \left(a_j^{(l,\lambda)} - \mu_{j\mathcal{B}}^{(l)} \right) = 0$$

となることを使った。続けて

$$\begin{aligned}
\frac{\partial s_{j\mathcal{B}}^{(l)}}{\partial a_j^{(l,i)}} &= \frac{\partial s_{j\mathcal{B}}^{(l)}}{\partial v_{j\mathcal{B}}^{(l)}} \frac{\partial v_{j\mathcal{B}}^{(l)}}{a_j^{(l,i)}} \\
&= \frac{1}{2\sqrt{v_{j\mathcal{B}}^{(l)} + \epsilon}} \frac{2}{\tilde{n}} \left(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)} \right) \\
&= \frac{1}{\tilde{n} s_{j\mathcal{B}}^{(l)}} \left(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)} \right) \qquad (12) \\
\frac{\partial \hat{a}_j^{(l,k)}}{\partial a_j^{(l,i)}} &= \frac{1}{(s_{j\mathcal{B}}^{(l)})^2} \left[s_{j\mathcal{B}}^{(l)} \frac{\partial}{a_j^{(l,i)}} \left(a_j^{(l,k)} - \mu_{j\mathcal{B}}^{(l)} \right) - \left(a_j^{(l,k)} - \mu_{j\mathcal{B}}^{(l)} \right) \frac{\partial s_{j\mathcal{B}}^{(l)}}{\partial a_j^{(l,i)}} \right] \\
&= \frac{1}{(s_{j\mathcal{B}}^{(l)})^2} \left[s_{j\mathcal{B}}^{(l)} \left(\delta_{ik} - \frac{1}{\tilde{n}} \right) - \frac{1}{\tilde{n} s_{j\mathcal{B}}^{(l)}} \left(a_j^{(l,k)} - \mu_{j\mathcal{B}}^{(l)} \right) \left(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)} \right) \right] \\
&= \frac{1}{s_{j\mathcal{B}}^{(l)}} \left[\delta_{ik} - \frac{1}{\tilde{n}} - \frac{1}{\tilde{n} (s_{j\mathcal{B}}^{(l)})^2} \left(a_j^{(l,k)} - \mu_{j\mathcal{B}}^{(l)} \right) \left(a_j^{(l,i)} - \mu_{j\mathcal{B}}^{(l)} \right) \right] \\
&= \frac{1}{s_{j\mathcal{B}}^{(l)}} \left(\delta_{ik} - \frac{1}{\tilde{n}} - \frac{\hat{a}_j^{(l,k)} \hat{a}_j^{(l,i)}}{\tilde{n}} \right) \qquad (13) \\
\frac{\partial y_j^{(l,k)}}{\partial a_j^{(l,i)}} &= \frac{\partial y_j^{(l,k)}}{\partial \hat{a}_j^{(l,k)}} \frac{\partial \hat{a}_j^{(l,k)}}{\partial a_j^{(l,i)}} = \gamma_j^{(l)} \frac{\partial \hat{a}_j^{(l,k)}}{\partial a_j^{(l,i)}} \\
&= \frac{\gamma_j^{(l)}}{s_{j\mathcal{B}}^{(l)}} \left(\delta_{ik} - \frac{1}{\tilde{n}} - \frac{\hat{a}_j^{(l,k)} \hat{a}_j^{(l,i)}}{\tilde{n}} \right) \qquad (14)
\end{aligned}$$

となる。

第 l 層の j 番目のユニットに対する損失関数 L は

$$L = L \left(y_j^{(l,k)}, \cdots, y_j^{(l,\tilde{n})} \right)$$

であり、かつ

$$y_j^{(l,i)} = y_j^{(l,i)} \left(a_j^{(l,1)}, \cdots, a_j^{(l,\tilde{n})} \right)$$

であるので、$a_j^{(l,i)}$ に対する勾配は以下のように求められる。

$$\begin{aligned}
\frac{\partial L}{\partial a_j^{(l,i)}} &= \sum_{\lambda=1}^{\tilde{n}} \frac{\partial L}{\partial y_j^{(l,\lambda)}} \frac{\partial y_j^{(l,\lambda)}}{\partial a_j^{(l,i)}} \\
&= \sum_{\lambda=1}^{\tilde{n}} \frac{\partial L}{\partial y_j^{(l,\lambda)}} \frac{\gamma_j^{(l)}}{s_{j\mathcal{B}}^{(l)}} \left(\delta_{i\lambda} - \frac{1}{\tilde{n}} - \frac{\hat{a}_j^{(l,\lambda)} \hat{a}_j^{(l,i)}}{\tilde{n}} \right) \\
&= \frac{\gamma_j^{(l)}}{s_{j\mathcal{B}}^{(l)}} \left[\frac{\partial L}{\partial y_j^{(l,i)}} - \frac{1}{\tilde{n}} \sum_{\lambda=1}^{\tilde{n}} \left(1 + \hat{a}_j^{(l,\lambda)} \hat{a}_j^{(l,i)} \right) \frac{\partial L}{\partial y_j^{(l,\lambda)}} \right] \qquad (15)
\end{aligned}$$

ここで $\partial L / \partial y_j^{(l,k)}$ 等は逆伝播によって出力層側から伝わってくるので、式 (15) によってバッチ正規化変換前の活性 $a_j^{(l,i)}$ に対する勾配が求まる。

次に、正規化パラメータ $\beta_j^{(l)}$, $\gamma_j^{(l)}$ については、式 (9) より

$$\frac{\partial y_j^{(l,k)}}{\partial \beta_j^{(l)}} = 1, \quad \frac{\partial y_j^{(l,k)}}{\partial \gamma_j^{(l)}} = \hat{a}_j^{(l,k)}$$

より

$$\frac{\partial L}{\partial \beta_j^{(l)}} = \sum_{\lambda=1}^{\tilde{n}} \frac{\partial L}{\partial y_j^{(l,\lambda)}} \frac{\partial y_j^{(l,\lambda)}}{\partial \beta_j^{(l)}} = \sum_{\lambda=1}^{\tilde{n}} \frac{\partial L}{\partial y_j^{(l,\lambda)}} \tag{16}$$

$$\frac{\partial L}{\partial \gamma_j^{(l)}} = \sum_{\lambda=1}^{\tilde{n}} \frac{\partial L}{\partial y_j^{(l,\lambda)}} \frac{\partial y_j^{(l,\lambda)}}{\partial \gamma_j^{(l)}} = \sum_{\lambda=1}^{\tilde{n}} \hat{a}_j^{(l,\lambda)} \frac{\partial L}{\partial y_j^{(l,\lambda)}} \tag{17}$$

となる。

式 (15) の勾配を使って重みパラメータを、式 (16)、(17) により正規化パラメータ $\beta_j^{(l)}$, $\gamma_j^{(l)}$ をそれぞれ更新する。

また、(16)、(17) を用いれば、式 (15) は以下のようにも表せる。

$$\frac{\partial L}{\partial a_j^{(l,i)}} = \frac{\gamma_j^{(l)}}{s_{j\mathcal{B}}^{(l)}} \left[\frac{\partial L}{\partial y_j^{(l,i)}} - \frac{1}{\tilde{n}} \left(\frac{\partial L}{\partial \beta_j^{(l)}} + \hat{a}_j^{(l,i)} \frac{\partial L}{\partial \gamma_j^{(l)}} \right) \right] \tag{18}$$

式 (18) より、バッチ正規化変換に対する誤差逆伝播法の流れは

- 式 (16)、(17) より $\dfrac{\partial L}{\partial \beta_j^{(l)}}$, $\dfrac{\partial L}{\partial \gamma_j^{(l)}}$ を計算する。
- 式 (18) より $\dfrac{\partial L}{\partial a_j^{(l,i)}}$ を計算する.

となる。

3. Python による実装

式 (5)-(9) の順伝播と式 (16)-(18) の逆伝播を Python で実装したプログラムを以下に示す。

```
class BatchNormalization :
    def __init__(self, gamma = 1.0, beta = 1.0):

        self.beta = beta
        self.gamma = gamma
        self.std = None
        self.ac = None
        self.y = None

        def forward(self, a):

            self.batch_size, self.layer_size = a.shape[0], a.shape[1]
            self.beta = self.beta*np.ones(self.layer_size)
            self.gamma = self.gamma*np.ones(self.layer_size)

            m = np.mean(a, axis = 0)
            v = np.mean((a-m)**2, axis = 0)
            std = np.sqrt(v + 10e-7)
            ac = (a - m)/std
            y = self.gamma*ac + self.beta
            self.std = std
            self.ac = ac

            return y

        def backward(self, dout):

            self.dbeta = dout.sum(axis = 0)
            self.dgamma = (self.ac*dout).sum(axis = 0)
            da = dout - (self.dbeta + self.ac*self.dgamma)/self.batch_size
            da *= self.gamma/self.std

            return da
```

ユニット数 $m = 10$、バッチサイズ $\tilde{n} = 20$ の入力に対して、損失関数を

$$L = \sum_{\lambda=1}^{\tilde{n}} \sum_{j=1}^{m} \left(y_j^{(\lambda)} \right)^2$$

とした場合における誤差逆伝播法と数値微分による勾配を比較する。このとき出力側から伝播してくる勾配は

$$\frac{\partial L}{\partial y_k^{(i)}} = 2y_k^{(i)}$$

となる。この計算のプログラムとその結果を以下に示す。

```
a = 0.1*np.random.rand(20,10) # 変換前の活性  サンプル数:、ユニット数:2010  e = 10e-8

# 誤差逆伝播法

BN = BatchNormalization(beta = 2, gamma = 2)
y = BN.forward(a)
dout = 2*y
da = BN.backward(dout)

print('誤差逆伝播法 :') print("dL/da[0,0] = ", da[0,0])
print("dL/beta[0] = ", BN.dbeta[0])
print("dL/dgamma[0] = ", BN.dgamma[0], '\n')

# 数値計算

a[0,0] += e y = BN.forward(a)
Loss1 = (y**2).sum()
a [0,0] -= 2*e
y = BN.forward(a)
Loss2 = (y**2).sum()
daa = (Loss1 - Loss2)/(2*e)
a[0,0] += e

BN.beta[0] += e
y = BN.forward(a)
Loss1 = (y**2).sum()
BN.beta[0] -= 2*e
y = BN.forward(a)
Loss2 = (y**2).sum()
dbb = (Loss1 - Loss2)/(2*e)
BN.beta[0] += e

BN.gamma[0] += e
y = BN.forward(a)
Loss1 = (y**2).sum()
BN.gamma[0] -= 2*e
y = BN.forward(a)
Loss2 = (y**2).sum()
dgg = (Loss1 - Loss2)/(2*e)
BN.gamma[0] += e

print('数値計算 :')
print("dL/da[0,0] = ", daa)
print("dL/dbeta[0] = ", dbb)
print("dL/dgamma[0] = ", dgg)

#---- 計算結果 -----誤差逆伝播法

 :
dL/da[0,0] =  0.0728064824102167
dL/beta[0] =  80.00000000000001
dL/dgamma[0] =  79.8903466071115数値計算

 :
dL/da[0,0] =  0.07280618774530012
dL/dbeta[0] =  80.0000009348878
dL/dgamma[0] =  79.89034543243179
```

誤差逆伝播法と数値微分による勾配の計算結果は、ほぼ一致している。このことからバッチ正規化層が正しく実装できていることが分かる。

4. 参考文献

1) 手塚 太郎 『しくみがわかる深層学習』 朝倉書店 (2018/6/25)

2) 竹縄 知之 『深層学習入門 6.3 バッチ正規化』 http://www2.kaiyodai.ac.jp/~takenawa/learning/