深層学習前編（day2）レポート

2024/6/16

1　Section1: 勾配消失問題

勾配消失問題とは、誤差逆伝搬法が下位層（入力層側）に進んでいくにつれて、連鎖律による乗算が多くなり、勾配が緩やかになることによって起きる現象。つまり、下位層のパラメータがほとんど更新されなくなってしまう。シグモイド関数は、微分した値の最大値は 0.25 であるため、勾配消失問題を引き起こしやすい。勾配消失問題の解決法として、以下の 3 点がある。

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

1.1　活性化関数の選択

シグモイド関数では勾配消失問題を引き起こしやすいため、別の活性化関数を使用するというアプローチ。具体的には、以下の式 1 で表される ReLU 関数がよく用いられる。

$$f(x) = \begin{cases} x(x > 0) \\ 0(x \leq 0) \end{cases} \tag{1}$$

特徴として、値が正のときは微分した値が 1 となるため、勾配消失問題を回避することができる。また、負のときは微分した値が 0 となるため、ニューラルネットワークのスパース化（不必要な重みを削除すること）に貢献できる。

1.2　重みの初期値設定

通常の、正規分布乱数に従った重みの初期値設定方法だと、下位層の勾配が 0 か 1 付近に固まる。そのため、初期値設定方法を変えることで勾配にバリエーションを持たせるアプローチ。以下の 2 つがよく用いられる。

1. Xavier
   シグモイド関数などの、S 字カーブ型の活性化関数に有効。重みの要素を、前の層のノード数の平方根で除算した値を初期値として用いる。

2. He
   ReLU 関数など、その他の活性化関数に有効。重みの要素を、前の層のノード数の平方根で除算した値に対し、さらに $\sqrt{2}$ をかけた値を初期値として用いる。

1.3 バッチ正規化

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アプローチ。一般的に、活性化関数に値を渡す前後に、以下の 4 つの処理を孕んだ層を加える。ここで、x_{ni} は入力値である。

1. ミニバッチ t 全体の平均を求める。

$$\mu_t = \frac{1}{N_t}\sum_{i=1}^{N_t} x_{ni} \tag{2}$$

2. ミニバッチ t 全体の標準偏差を求める。

$$\sigma_t{}^2 = \frac{1}{N_t}\sum_{i=1}^{N_t}(x_{ni} - \mu_t)^2 \tag{3}$$

3. 入力値に対して 0 を中心とする正規化を行う。ここで θ はゼロ乗算を防ぐための微小な値。

$$\hat{x}_{ni} = \frac{x_{ni} - \mu_t}{\sqrt{\sigma_t{}^2 + \theta}} \tag{4}$$

4. 正規化した値で出力値を求める。ここで γ と β はそれぞれスケーリングパラメータとシフトパラメータであり、誤差逆伝搬法と確率的勾配降下法によって学習される。一般的には、この出力値を活性化関数の計算に用いる。

$$y_{ni} = \gamma x_{ni} + \beta \tag{5}$$

1.4 実装演習

1.4.1 2_2_2_vanishing_gradient_modified.ipynb

mnist データを用いて、活性化関数、重みの初期値設定を変えてみる。
図 1 は、活性関数にシグモイド、重みの初期値設定に正規分布による乱数を用いた正答率の推移である。訓練・テストともに学習がうまく進んでいない。図 2 は、活性関数に ReLU、重みの初期値設定に正規分布による乱数を用いた正答率の推移である。初期値の設定が悪いため、最初は学習がうまく進んでいない。図 3 は、活性関数にシグモイド、重みの初期値設定に Xavier を用いた正答率の推移である。訓練・テストともに学習が順調に進んでいる。図 4 は、活性関数に ReLU、重みの初期値設定に He を用いた正答率の推移である。訓練・テストともに学習が順調に進んでいる。

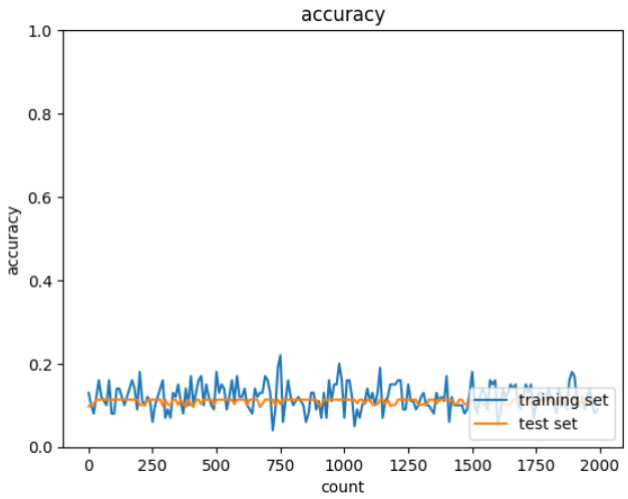

図 1: 活性化関数：シグモイド、初期値設定：正規分布

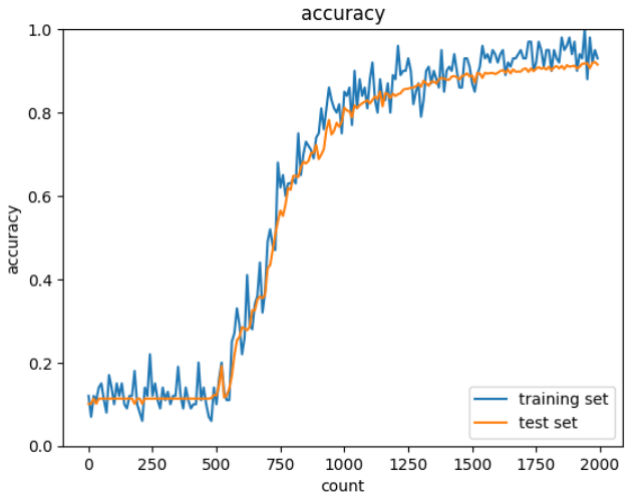

図 2: 活性化関数：ReLU、初期値設定：正規分布

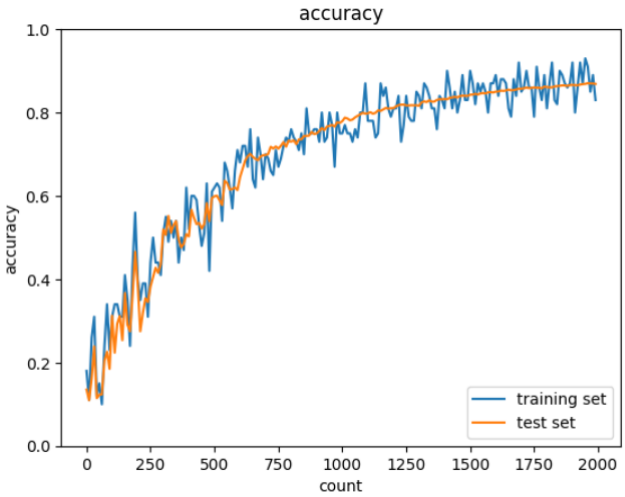

図 3: 活性化関数：シグモイド、初期値設定：Xavier

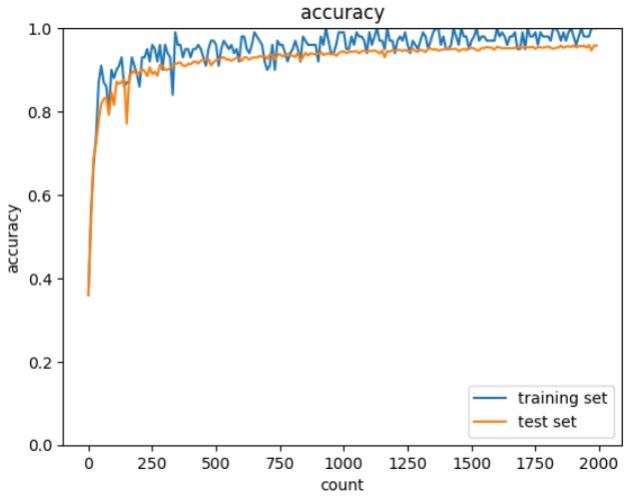

図 4: 活性化関数：ReLU、初期値設定：He

1.4.2 2_3_batch_normalization.ipynb

mnist データを用いて、バッチ正規化を行ってみる。図 5 は、バッチ正規化を用いたときの正答率の推移である。活性化関数に ReLU、重みの初期値設定に He を用いたときのような結果となった。しかし、その結果よりも、活性化関数や初期値設定の選択が悪いため、正答率の向上速度は遅い。

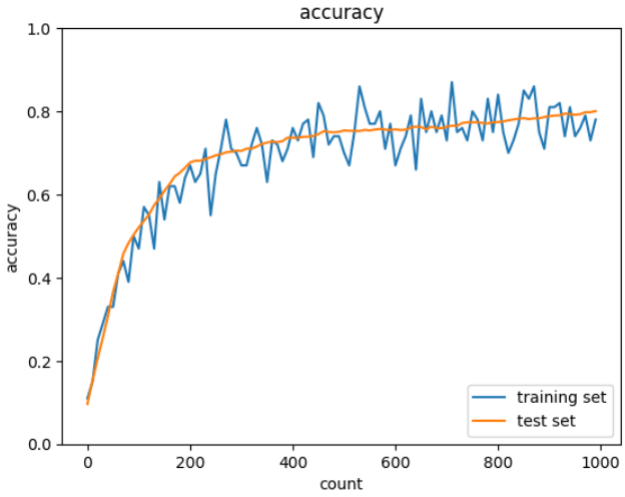

図 5: バッチ正規化

1.5 確認テスト

1.5.1 連鎖律の原理を使い、dz/dx を求めよ。

$$z = t^2$$
$$t = x + y$$

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}x} = 2t = 2(x + y)$$

1.5.2 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

(2)

1.5.3 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。

重みを 0 で初期化すると、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もつ意味がなくなり、正しい学習が行えなくなる。

1.5.4　一般的に考えられるバッチ正規化の効果を 2 点挙げよ。

1. 中間層の重みの更新が安定化される。結果として学習が速く進む。
2. 学習データの極端なばらつきを抑えられ、過学習を抑えられる。

2　Section2: 学習率最適化手法

勾配降下法の誤差関数 $E(\boldsymbol{w})$ に乗算する値である学習率は、値が大きすぎる場合は最適解にたどり着けず発散してしまう。一方で値が小さすぎる場合は退位的最適解を見つけられず局所最適解に陥ってしまう。そのため、学習率を最適化する手法がいくつか提案されている。代表的なのは以下の 4 つである。

1. モメンタム
   式 6,7 で表される、誤差をパラメータで微分したものと学習率の積を減算した後、現在の重みに前回の重みを減算した値を慣性 μ の積を加算する。メリットとしては、大域的最適解を見つけやすいことと、重みの更新量を使うため、谷間についてから最適値にいくまでの時間が早い。

$$V_t = \mu V_{t-1} - \epsilon \nabla E \tag{6}$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + V_t \tag{7}$$

2. AdaGrad
   式 8,9,10 で表される、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現在の重みから減算する。メリットとしては、勾配の緩やかな斜面に対して最適値に近づけることができる。一方で、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。

$$h_0 = \theta \tag{8}$$

$$h_t = h_{t-1} + (\nabla E)^2 \tag{9}$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E \tag{10}$$

3. RMSProp
   式 11,12 で表される、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現在の重みから減算する。AdaGrad の改良版。メリットとして、大域的最適解を見つけやすく、ハイパーパラメータ θ,α の調整が必要な場合が少ない。

$$h_t = \alpha h_{t-1} + (1-\alpha)(\nabla E)^2 \tag{11}$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E \tag{12}$$

4. Adam
   モメンタムの、過去の勾配の指数関数的減衰平均と、RMSProp の、過去の勾配の 2 乗の指数関数的減衰平均それぞれを孕んだ最適化アルゴリズムである。モメンタムおよび RMSProp のメリットを孕んだアルゴリズムとなっている。

2.1 実装演習

2.1.1 2_4_optimizer.ipynb

mnist データを用いて、学習率最適化手法による違いをみてみる。図 6,7 はそれぞれモメンタム、AdaGrad による正答率の推移である。これら 2 つは学習がうまく進んでいないため、活性化関数や重みの初期値設定を返れば改善する可能性がある。一方で、図 8,9 から、RMSProp、Adam では学習がうまく進んでいるため、mnist で鞍点問題が起きているかもしれない。

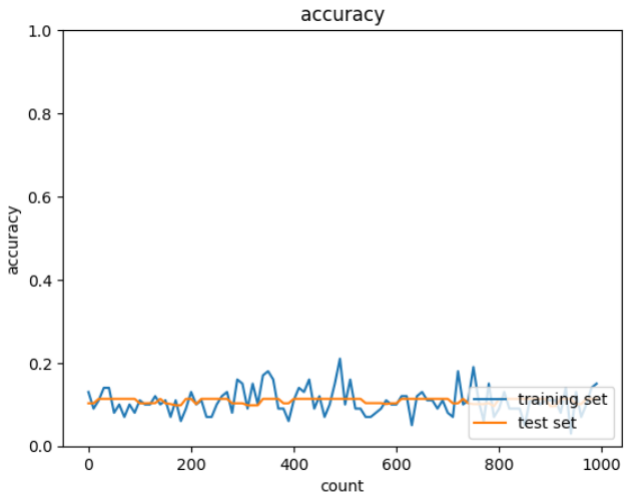

図 6: モメンタム

2.2 確認テスト

2.2.1 モメンタム・AdaGrad・RMSProp の特徴をそれぞれ簡潔に説明せよ。

- モメンタム：大域的最適解を見つけやすい。重みが大きく更新されるとその分更新量も多くなるため、谷間についてから最も低い位置（最適解）にいくまでの時間が早い。
- AdaGrad：勾配の緩やかな斜面に対して、最適解を見つけやすい。一方で、学習率が徐々に小さくなるので、鞍点問題（勾配が 0 となってしまう点で学習が止まってしまう）を引き起こすことがある。
- RMSProp：大域的最適解を見つけやすい。ハイパーパラメータの調整が必要な場合が少ない。

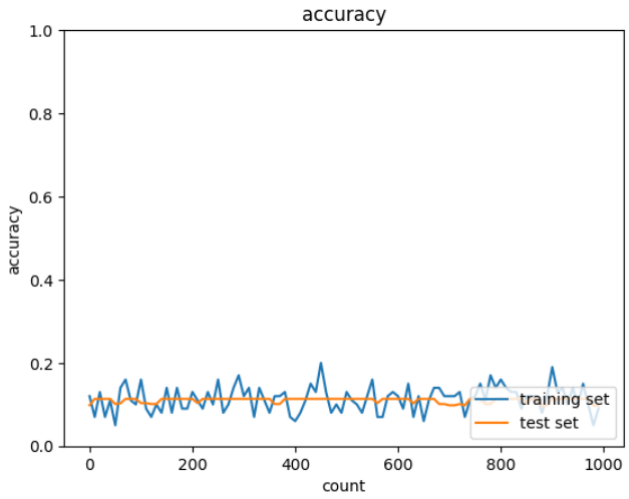

図 7: AdaGrad

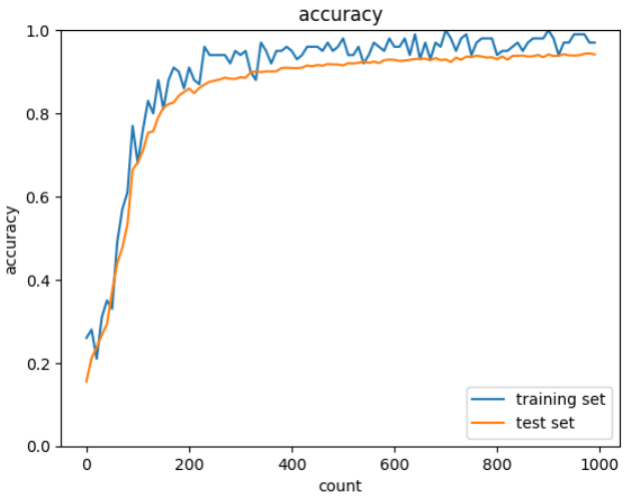

図 8: RMSProp

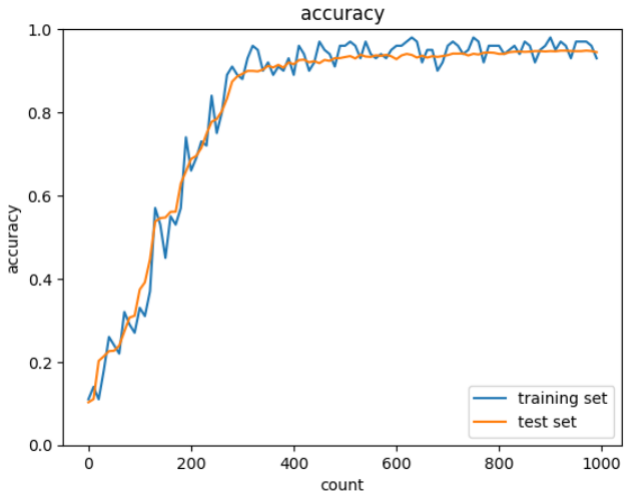

図 9: Adam

3 Section3: 過学習

過学習とは、テスト誤差と訓練誤差とで学習曲線が乖離してしまうこと。訓練用データに対して過剰にフィットしたモデルを作ってしまい、汎化性能を落としてしまう。原因としては、学習回数が多い、ネットワークの自由度（層数、ノード数、パラメータの値など）が高いことが挙げられる。過学習を防ぐ手法として、以下の正則化手法が挙げられる。

1. L1 正則化、L2 正則化
2. ドロップアウト

3.1 L1 正則化、L2 正則化

一部の重みが大きすぎる値をとることで過学習が発生することに着目した手法である。具体的には、誤差に対して正則化項を加算することで、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る。正則化項として、式 13 の p ノルムが用いられる。

$$\|x\|_p = (|x_1|^p + \cdot + |x_n|^p)^{\frac{1}{p}} \tag{13}$$

$p = 1$ で誤差関数に加算した場合は L1 正則化、$p = 2$ で加算した場合は L2 正則化と呼ぶ。L1 正則化は、一部の重みが 0 となる点が最適値となりやすく、パラメータ事態を削減することができる。L2 正則化は、最適値をとる重みから一定量ずれた値が最適値となりやすく、パラメータが発散することを抑制できる。

3.2 ドロップアウト

ノードの数が多いことで過学習が発生することに着目した手法である。具体的には、ランダムにノードを削除して学習させる。メリットとしては、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と解釈できる。一方で、自由度が落ちるため、誤差の下がり方は遅くなる。

3.3 実装演習

3.3.1 2_5_overfiting.ipynb

図 10 は、mnist データを用いてエポック数 1000 で学習したときの正答率の推移である。訓練データに対してはほぼ正答しているが、その分テストデータに対しては正答率が悪く、汎化性能が落ちている。また、重みに 0.1 をかけて大きくならないように抑制してみる（図 11）と、テストデータに対する正答率はよくなっているが、それでも 7 割ほどの正答率しかない。
そこで正則化手法として L1 正則化を用いる（図 12）と、一部の重みの学習が進まなくなった影響か、正答率が上下した結果となり、これでも過学習の傾向が見られる。一方でドロップアウトを用いる（図 13）、またはドロップアウトと L1 正則化を用いると、学習が遅くなったが、訓練データとテストデータそれぞれの正答率の差が小さくなったため、このまま学習を進めていけばテストデータに対する正答率がさらによくなる可能性がある。そのため、過学習を抑えられているといえる。

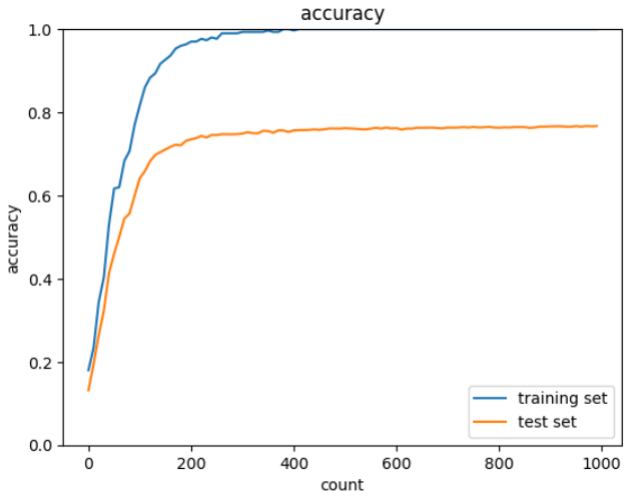

図 10: エポック数 1000 での学習結果（過学習）

3.4 確認テスト

3.4.1 機械学習で行われる線形モデル（線型回帰、主成分分析...etc）の正則化は、モデルの重みを制限することで可能となる。前述の線形モデルの正則化手法の中にリッジ回帰という手法があり、その特徴として正しいものを選択しなさい。

(a)

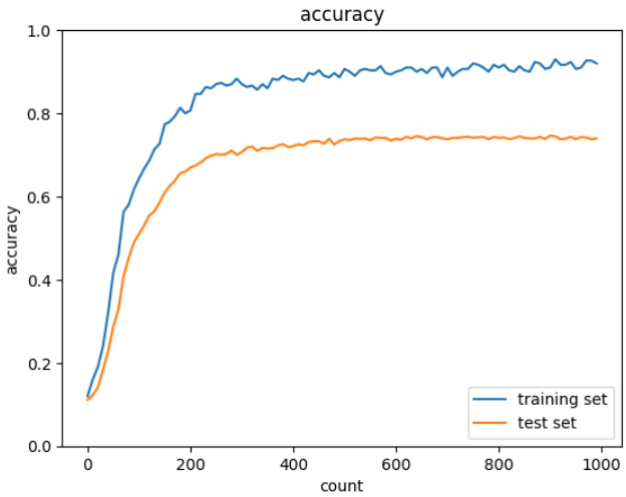

図 11: 重みに 0.1 をかけて学習した結果

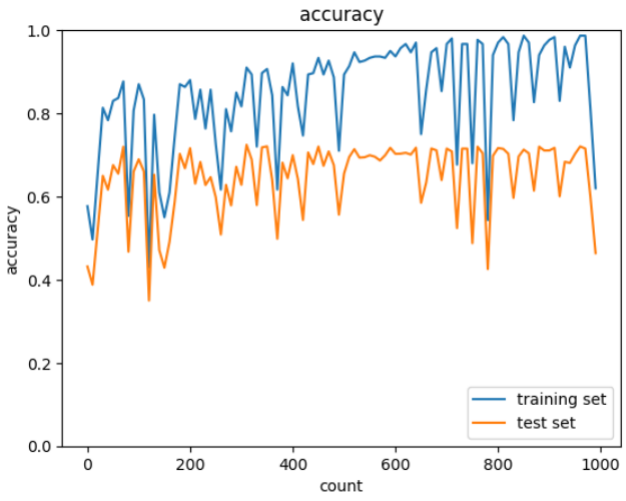

図 12: L1 正則化を用いた学習結果

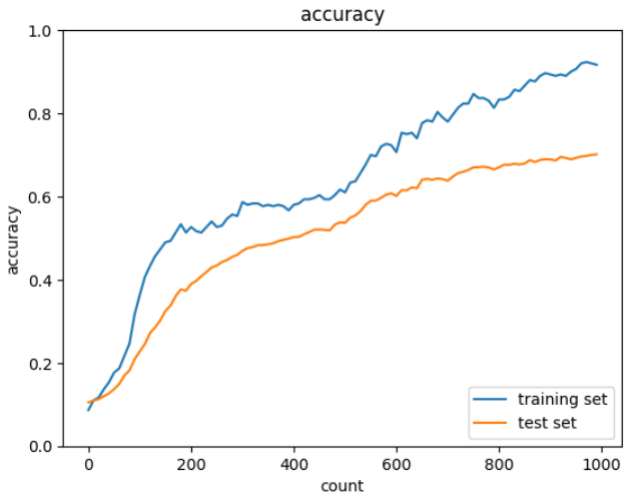

図 13: ドロップアウトを用いた学習結果

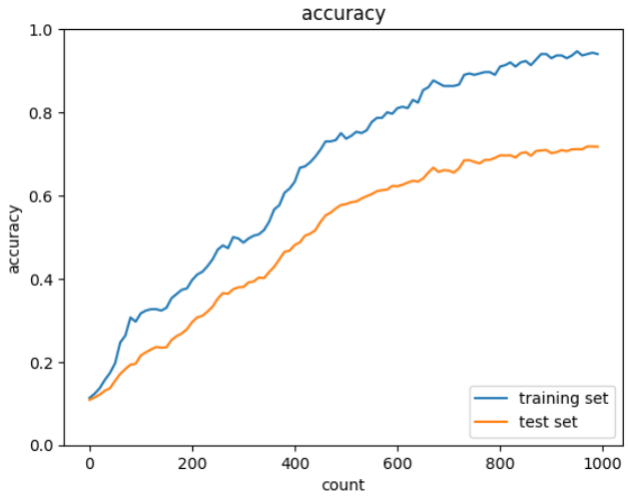

図 14: ドロップアウトと L1 正則化を用いた学習結果

3.4.2　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。

右図

4　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（CNN）とは、畳み込み層、プーリング層、全結合層を組み合わせて、画像などの次元間で繋がりのあるデータを処理できるニューラルネットワークである。CNN の構造例を図 15 に示す。畳み込み層では、フィ

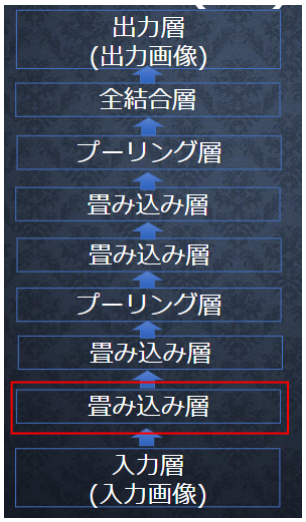

図 15: CNN の構造例

ルター（これまでのニューラルネットワークでいう重み）を用いて、周囲の値を考慮した演算を行う。そのため、画像などのように緑の点の近くは緑の点である可能性が高いのような、繋がりのあるデータとして学習を行える。以下 4 つの概念が結果に影響してくる。

1. バイアス
   フィルターで計算後に演算する値。
2. パディング
   入力データのサイズをどれだけ大きくして畳み込み演算するかを決める値。通常、畳み込み演算を繰り返し行うとサイズが小さくなっていくため、それを抑制できる。
3. ストライド
   フィルターを何個飛ばしでずらして畳み込み演算するかを決める値。
4. チャンネル
   いくつのフィルターを用いて、1 つ（あるいは複数）のデータに対して畳み込み演算をするかを決める値。

プーリング層では、フィルタを使わず、対象領域の最大値または平均値を取得することでデータサイズを小さくしていく。
全結合層では、通常のニューラルネットワークを用いて、全てのノード（入力値）を結合し 1 次元データに変換する。結合するため、次元間の繋がりを考慮しないデータとして扱う点に注意が必要である。

4.1 実装演習

4.1.1 2_6_simple_convolution_network.ipynb

図 16 は、mnist データに対して、「conv-relu-pool-affine-relu-affine-softmax」の順に処理を行う CNN で学習させた結果である。シンプルな CNN 構造であるが、それでもテストデータに対する正答率は 9 割を超えている。層を深くすることでさらに良くなる可能性があるが、増やしすぎると 1 回の実行に時間がかかることや、過学習が起きる可能性があるため、CNN を実行するには並列で動かすか計算速度の速いコンピュータを用いたほうがよい。

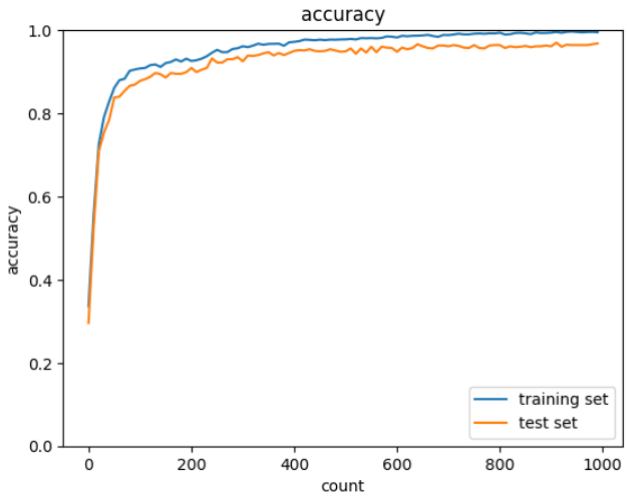

図 16: CNN での実行結果

4.2 確認テスト

4.2.1 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。

高さ：$(6 + 2 \times 1 - 2)/1 + 1 = 7$
横：$(6 + 2 \times 1 - 2)/1 + 1 = 7$
よって、7×7

5 Section5: 最新の CNN

最新の CNN として AlexNet が挙げられる。AlexNet は、2012 年の画像分類の精度を競う大会で優勝した CNN である。図 17 に、AlexNet の構造を示す。AlexNet は、3 つの畳み込み層、2 つのプーリング層、3 つの全結合層で構成されている。また、過学習を防ぐために、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。

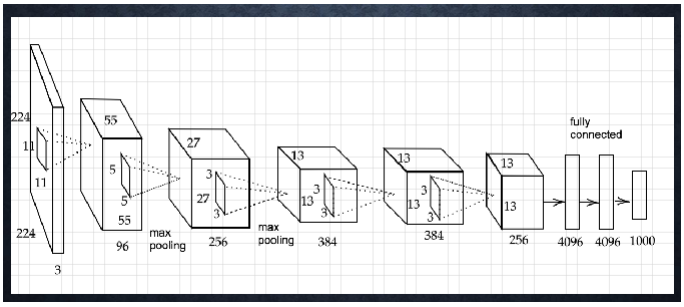

図 17: AlexNet の構造

6 [フレームワーク演習] 正則化/最適化

ここでは、Elastic Net、正規化レイヤーについて説明する。

6.1 Elastic Net

3 章で説明したように、過学習を防ぐ手法として、L1 正則化、L2 正則化がある。これらを組み合わせたものとして、式 14 で表される Elastic Net がある。ここで、λ_1,λ_2 は各正則化項の制約の強さを調整するハイパーパラメータである。

$$ElasticNet = E(\boldsymbol{w}) + \lambda_1 \sum_{j=1}^{m} w_j^2 - \lambda_2 \sum_{j=1}^{m} |w_j| \tag{14}$$

6.2 正規化レイヤー

1 章で説明したバッチ正規化も、過学習を抑える効果を持つ。バッチ正規化はミニバッチ単位かつ各チャンネルごとに正規化を行うが、他にどの単位で正規化を行うかで以下の 2 つの方法がある。

1. レイヤー正規化
   N 個のサンプルのうち 1 つに着目し、チャンネルを含めた全ての値を正規化の単位とする。ミニバッチの数に依存しないため、バッチ正規化の問題（ミニバッチのサイズを大きくとれない場合は効果が薄くなってしまう）問題を解消できていると考えられる。

2. インスタンス正規化
   各サンプルの各チャネルごとに正規化を行う。バッチ正規化におけるバッチサイズが 1 の場合と同じ処理である。

6.3 実装演習

6.3.1 2_9_regularization.ipynb

図 18,19,20,21 は、cifar10 データに対して、正則化項なし、L1、L2、ElastiNet をぞれぞれ適用した CNN で学習させた結果である。ただし、横軸はエポック数、縦軸は誤差である。これらの図から、Elastic Net は正則化項なし・L1 正則化・L2 正則化よりもテストデータに対する誤差が小さく、より優秀であることがわかる。

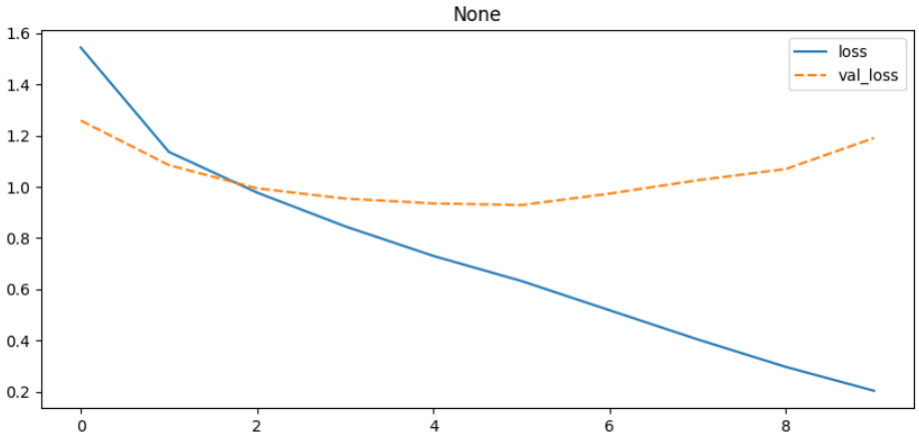

図 18: 正則化項なしの CNN の学習結果

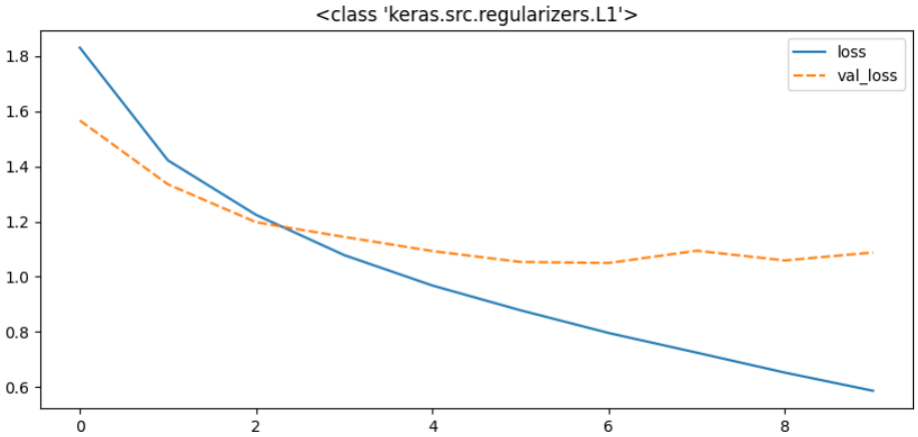

図 19: L1 正則化を用いた CNN の学習結果

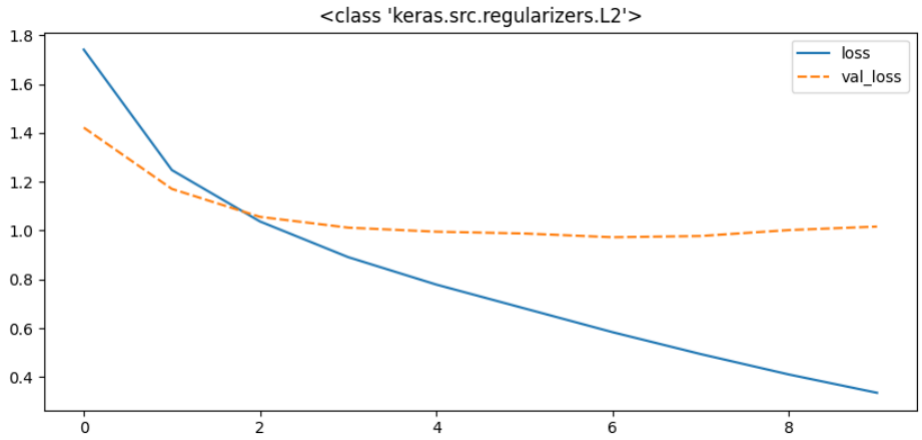

図 20: L2 正則化を用いた CNN の学習結果

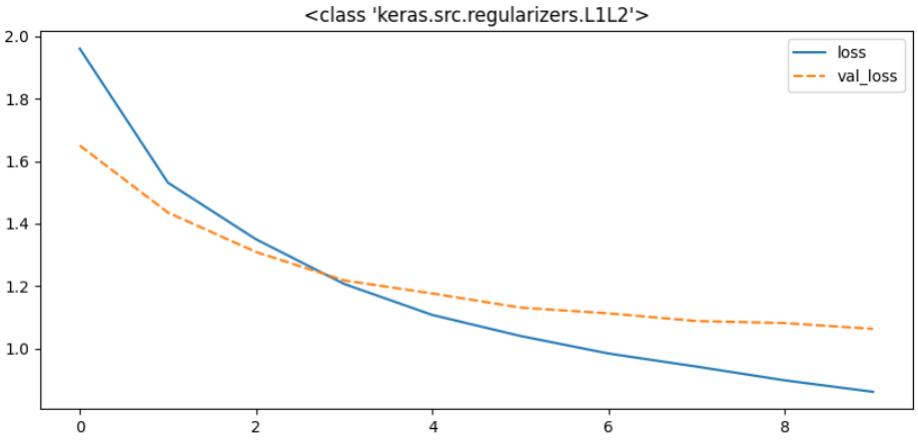

図 21: Elastic Net を用いた CNN の学習結果

6.3.2 2_10_layer-normalization.ipynb

図 22,23,24,25 は、cifar10 データに対して、正規化なし、バッチ正規化（バッチサイズ 256）、レイヤー正規化、インスタンス正規化をぞれぞれ適用した CNN で学習させた結果である。ただし、横軸はエポック数、縦軸は誤差である。これらの図から、どの手法でもエポック数が増えると過学習の傾向が見られるが、バッチ正規化またはインスタンス正規化を用いた CNN は、他の手法に比べてテストデータに対する誤差が大きくなることを抑制している。

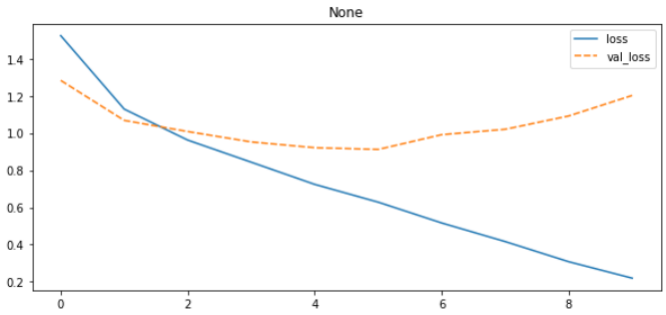

図 22: 正規化なしの CNN の学習結果

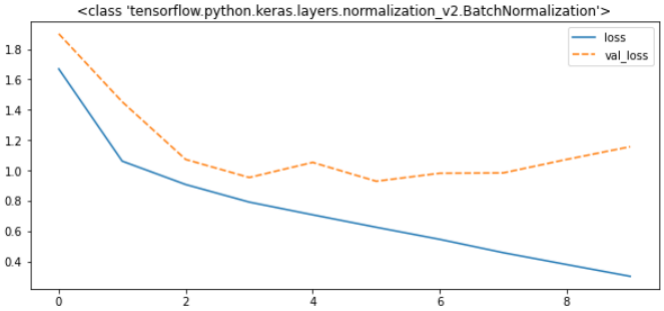

図 23: バッチ正規化を用いた CNN の学習結果（バッチサイズ 256）

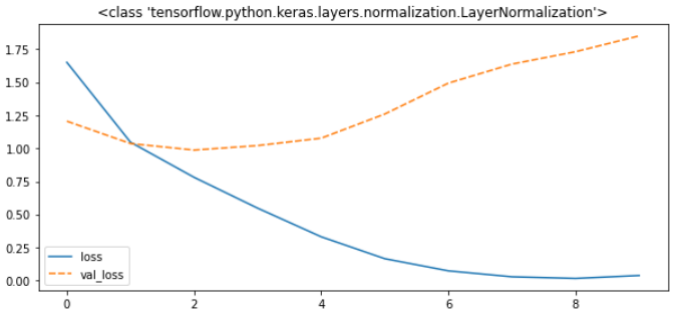

図 24: レイヤー正規化を用いた CNN の学習結果

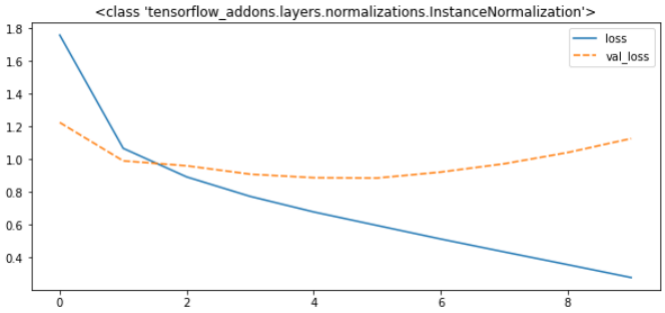

図 25: インスタンス正規化を用いた CNN の学習結果