Rabbit Challenge 深層学習\_day1day2レポート

1. **深層学習day1**
   1. **Section1：入力層～中間層**

* **要約**

〇入力層～中間層

特記事項無し（これまでの学習の復習）

〇入力層の設計

取りうるデータは、実数、確率、フラグ値が挙げられる。入力層として問題があるデータは以下の通り。

・欠損値が多い

・誤差が大きい

・出力そのもの、出力を加工した情報(偏見が入る)

・連続性のないデータ(背番号など)

・無意味な数が割り当てられているデータ

　悪い例：Yes：1、No：0、どちらでもない：-1、無回答：-1

　良い例：Yes：1、No：-1、どちらでもない：0、無回答：無

　⇒カテゴリ変数ではなく、one-hotベクトルの方が望ましい

入力層の加工として、以下が挙げられる。

・欠損値の対応

対応として、補完(ゼロ、平均値、中央値など)、除外(欠損値を含むレコードを除外、欠損値を含む説明変数を除外)

・正規化・正則化

説明変数の尺度による影響を無効化するため、数値の正規化(0～1の値にするため最大値で割る)、正則化(平均0、分散1にする)がある

* **実装演習結果(キャプチャーorサマリと考察)**

〇1\_1\_forward\_propagation.ipynb

・順伝播（単層・単ユニット）

入力層から中間層への計算を確認。入力x、重みw、バイアスbの型は、x:(2,)、w(2,1)、b(1,)であり、中間層は(1,)。重み、バイアスの値を変化させ、動作確認を行った。また、手計算を行い、計算方法を再確認した。

* **確認テストなど、自身の考察結果**

入力x、重みw、バイアスbの行列の型について、以下の通り理解した。

　x(訓練データレコード数、属性数)

　w(xの属性数、次の中間層の属性数)

　b(訓練データレコード数、次の中間層の属性数)

* 1. **Section2：活性化関数**
* **要約**

活性化関数により、出力の強弱をつけることが可能となる。

中間層用の活性化関数として、ReLU関数、シグモイド関数、ステップ関数が挙げられる。

・ステップ関数

しきい値を超えたら発火する関数で0、1をとる。線形分離可能なものしか学習できないことが課題。

・シグモイド関数

0～1の間をとり、信号の強弱を表現できる。勾配消失問題を引き起こすことが課題。

・ReLU関数

勾配消失問題の回避とスパース化に貢献し、よい成果が得られる。

* **実装演習結果(キャプチャーorサマリと考察)**

〇1\_1\_forward\_propagation.ipynb

・順伝播（単層・複数ユニット）について、

入力層から中間層への計算を確認。入力x、重みw、バイアスbの型は、x:(4,)、w(3,4)、b(3,)であり、中間層は(3,)。中間層の活性化関数にシグモイド関数を利用。行列の内積がdot(w,x)で計算されており混乱した(dow(x,w)との混在)が、行列の型として問題ないことを確認した。

* **確認テストなど、自身の考察結果**

〇線形と非線形の違いを図視化し、簡単な説明

線形関数の完結な説明として、「加法性、斉次(せいじ)性を満たす」という表現を確認。

〇順伝播（3層・複数ユニット）で活性化関数を確認

ReLU関数がnp.maximum(0,入力ベクトル)で表現できることを確認した。

* **演習問題、参考図書、修了課題などの関連レポート**

活性化関数の種類を確認するため、以下のサイトを参考にした。

https://qiita.com/kuroitu/items/73cd401afd463a78115a

* 1. **Section3：出力層**
* **要約**

〇誤差関数

誤差関数は、以下の通り使い分ける。

　分類の場合：クロスエントロピー誤差

　回帰の場合：平均二乗誤差

〇活性化関数

活性化関数は中間層と出力層で目的が異なる。

　中間層：信号の強弱を調整

　出力層：信号の大きさをそのまま変換(比例)。分類問題の場合、出力を確率とする(0～1の値をとり、総和を1とする)

出力層用の活性化関数は、ソフトマックス関数、恒等社増、シグモイド関数が挙げられる。

　回帰の場合：恒等写像

　分類の場合：(二値)シグモイド関数、(多クラス)：ソフトマックス関数

* **実装演習結果(キャプチャーorサマリと考察)**

〇1\_1\_forward\_propagation.ipynb

・順伝播（3層・複数ユニット）

入力層、中間層2つ、出力層への計算を確認。中間層の活性化関数にrelu関数、出力層の活性化関数に恒等関数を利用。

・多クラス分類（2-3-4ネットワーク）

入力層サイズ、中間層サイズ、出力層サイズを調整し、2-3-4のネットワークを3-5-6に変更。中間層の活性化関数にrelu関数、出力層の活性化関数にsoftmax関数を利用。多クラス分類であるため、誤差関数に交差エントロピーを利用。

・回帰（2-3-2ネットワーク）

入力層サイズ、中間層サイズ、出力層サイズを調整し、2-3-2のネットワークを3-5-4に変更。中間層の活性化関数にrelu関数、出力層の活性化関数に恒等関数を利用。回帰であるため、誤差関数に平均二乗誤差を利用。

・2値分類（2-3-1ネットワーク）

2-3-1のネットワークを、5-10-20-1に変更。

中間層を1つ追加し、各層のshapeを変更した。

中間層の活性化関数にrelu関数、及びシグモイド関数、出力層の活性化関数に恒等関数を利用。二値分類であるため、誤差関数に交差エントロピーを利用。

* **確認テストなど、自身の考察結果**

・誤差関数を2乗する理由

誤差を整数で表現するため

・2で割る理由

誤差逆伝播時に誤差関数(二乗誤差)の微分するが、その計算を簡単にするため（本質的な理由はない）

・ソフトマックス関数のコード説明

当動画の説明では理解が困難。1次元配列は、参考サイトにより計算の流れがイメージできた。

・交差エントロピーのコード説明

交差エントロピーの-Σdi log yiの数式に対し、プログラム上でdiがどのように表現されているのか理解できなかった。実際にスモールプログラムで説明してほしい。

* **演習問題、参考図書、修了課題などの関連レポート**

ソフトマックス関数の理解を深めるため、以下のサイトを参考にソースの確認を行った。

https://qiita.com/shohei-ojs/items/d66783bcead3eb7efd82

* 1. **Section4：勾配降下法**
* **要約**

誤差を最小化するパラメータを発見する手法。誤差関数をwで偏微分と学習率の積を用いて、wを更新する。

・学習率が大きすぎた場合

最小値にいつまでもたどり着かず発散してしまう。

・学習率が小さい場合、発散することはないが、収束するまでに時間かかる。また、極値で学習が終わってしまう可能性がある。勾配降下法の学習率の決定・収束性向上のためのアルゴリズムとして、Momentum、AdaGrad、Adadelta、Adamがある。

勾配降下法の種類として、以下がある。

　①確率的勾配降下法(SGD)

　②ミニバッチ勾配降下法

①はランダムに抽出したサンプルを用い、メリットは以下の3点。

・データが冗長な場合の計算コストの軽減

・望まない極小値に収束するリスクを軽減

・オンライン学習ができる

②は、ランダムに分割したデータの集合(ミニバッチ)を用い、メリットは以下の通り。

・確率的勾配降下法のメリットを損なわず、計算機の計算資源を有効利用(スレッド並列化、GPUを利用したSIMD並列化など)できる

* **実装演習結果(キャプチャーorサマリと考察)**

〇1\_3\_stochastic\_gradient\_descent.ipynb

10万件のデータの中から、epochを1000件としてランダムにデータを抽出。1000回の学習で誤差がどのように下がっているかを可視化。

* **確認テストなど、自身の考察結果**

・オンライン学習とは

学習データを入力するたびに都度パラメータを更新し、学習を進めていく手法であることを確認。十分なバッチ学習を行うにはパソコン性能では限界があり、通例オンライン学習を用いる。また、バッチ学習でモデル生成後、オンライン学習を行うことで再学習が不要になると考える。

* **演習問題、参考図書、修了課題などの関連レポート**

勾配降下法のアルゴリズムを以下のサイトで再確認した。

https://postd.cc/optimizing-gradient-descent/

* 1. **Section5：誤差逆伝播法**
* **要約**

誤差勾配の計算方法として、数値微分はプログラムで微小な数値を生成し、微分を計算する一般的な手法。各パラメータwそれぞれについて順伝播の計算を繰り返し行う必要があり、計算負荷が大きい。そこで、誤差逆伝播方を用いる。算出された誤差を出力パラメータから順に微分し、前の層へと伝播。最小限の計算で各パラメータでの微分値を解析的に計算する手法。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(微分の連鎖律を理由)。サンプルコードの誤差逆伝播の計算を確認した。

* **実装演習結果(キャプチャーorサマリと考察)**

〇1\_2\_back\_propagation.ipynb

2-3-2ネットワークであり、中間層の活性化関数にrelu関数、出力層の活性化関数にsoftmax関数を利用。誤差関数はクロスエントロピーを用い、学習率は0.01。

誤差逆伝播として、損失関数をyで偏微分し、Δ2を計算。※ソース誤り(sigmoidの導関数を使っている)

b2に対し、Δ2をb2で偏微分し、勾配を計算。

w2に対し、Δ2をw2で偏微分し、勾配を計算。

微分のチェインルールより中間層のrelu関数を偏微分してΔ1を計算。

b1に対し、Δ1をb1で偏微分し、勾配を計算。

w1に対し、Δ1をw1で偏微分し、勾配を計算。

* **確認テストなど、自身の考察結果**

〇誤差逆伝播法における、既に行った計算結果を保持し、流量しているソースコードを抽出。微分の連鎖律の通りに計算が行われ、その値が誤差逆伝播時に流用されていることを確認した。

1. **深層学習day2**
   1. **Section1：Section1：勾配消失問題**

* **要約**

誤差逆伝播法が入力層に進んでいくにつれて、勾配が小さくなっていく。そのため、勾配降下法による更新では、入力層側のパラメータはほとんど更新されず、訓練は最適に収束しなくなる。多くの活性化関数では偏微分値が0～1の数値をとるものが多い。入力層のパラメータ更新は、微分のチェインルールより、偏微分値の掛け算で表現されるが、上記の理由から勾配消失が起こる。勾配消失の対策として、以下が挙げられる。

　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について、

ReLU関数の微分値は、以下の通りとなり、勾配消失が発生しない上、重みのスパース化(0/1でわかりやすいモデルの構築が可能)が期待できる。

　・x<0の場合、微分値は0

　・その他の場合、微分値は1

②について、

広い観点でモデルを作成するために、重みやバイアスの初期値は乱数が使用される。

従来は、平均0、分散1の標準正規分布が利用されていたが、以下の問題点があった。

・活性化関数にシグモイド関数を利用している場合、出力は0、もしくは1に偏る

・活性化関数にReLu関数を利用している場合、出力は0に偏る

上記の場合、微分値が0となり、勾配消失が発生する。

そこで、活性化関数によって以下の通り初期値を使い分けることで、活性化関数の表現を保ったまま勾配消失の対策ができる。

・活性化関数にシグモイド関数を利用している場合、Xavier初期値

・活性化関数にReLu関数を利用している場合、He初期値

③について、

データセットは学習用、検証用データに分割する。そのうち学習用データはミニバッチに分割する。

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であり、活性化関数に値を渡す前後にバッチ正規化の処理を持つ層を加える。それにより、過学習を抑え、学習が収束しやすくなるメリットがある。

* **実装演習結果(キャプチャーorサマリと考察)**

〇2\_2\_2\_vanishing\_gradient\_modified.ipynb

以降、mnistデータを利用。

・sigmoid-gauss

活性化関数にsigmoid関数、重み初期化の標準偏差を0.01とした場合、学習が行われないことを確認（2000回の学習で精度が20%以下）

・ReLU-gauss

活性化関数にReLU関数、重み初期化の標準偏差を0.01とした場合、学習が行われることを確認(2000回の学習で91%程度)。ただし、約500回程度の学習では精度20%程度と低く、学習効率が低い。

・sigmoid - Xavier

活性化関数にsigmoid関数、重み初期化をXavierとした場合、学習が行われることを確認(2000回の学習で88%程度)。1つ目のsigmoid-gaussと比較し、重み初期値を変更するだけで学習が行われることを確認。

・ReLU - He

活性化関数にReLU関数、重み初期化をHeとした場合、学習が行われることを確認(2000回の学習で95%程度)。学習速度も精度も最も高くなった。

〇2\_3\_batch\_normalization.ipynb

バッチ正規化を行わない場合、学習が行われないことを確認（1000回の学習で精度が20%程度）。一方、バッチ正規化を行った場合、学習が行われることを確認（1000回の学習で精度が83%程度）。

* **確認テストなど、自身の考察結果**

〇微分連鎖律の確認

問題なく回答できた。

〇シグモイド関数を微分した場合の最大値

最大0.25で0以上の値をとる。

中間層が多い場合、シグモイド関数を活性化関数に利用すると勾配消失が発生することを確認できた。

〇重みの初期値に0を設定するとどうなるか。

すべての重みｎ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。（正しい学習が行えない）

* **演習問題、参考図書、修了課題などの関連レポート**

重み初期値の設定について、以下のサイトを参考に追加学習を行った。

https://ai-trend.jp/basic-study/neural-network/initial\_value/

バッチ正規化について、以下のサイトを参考に追加学習を行った。

https://qiita.com/jun40vn/items/2105467cea35f179ea45

* 1. **Section2：学習率最適化手法**
* **要約**

学習率最適化について、主要は手法は4種類挙げられる。

　①モメンタム

　②AdaGrad

　③RMSProp

　④Adam（現在利用する手法）

学習率が大き過ぎる場合、最適値にいつまでも辿り着かず発散する。

学習率が小さ過ぎる場合、収束までに時間がかかったり、極値に陥り学習が止まる可能性がある

そこで、学習率設定について、以下の設定方針を取る

・初期の学習率は大きく、徐々に小さくする

・パラメータ毎に学習率を変化させる

①について、

前回更新量と慣性パラメータの積から勾配降下法による更新値を引いた値を、今回の重み更新量とする方式。大域的最適解になりやすく、最適化までの収束時間が早いメリットがある。

②について、

勾配が緩やかな斜面に対して最適解に近づけるが、学習率が徐々に小さくなるので鞍点(あんてん)問題が発生する場合がある。

③について、

AdaGradの問題を解消した手法であり、大域最適解となりやすく、ハイパーパラメータθの調整が容易である。

④について、

以下の含んだ最適化手法である。

・モメンタムの過去勾配の指数関数的減衰平均

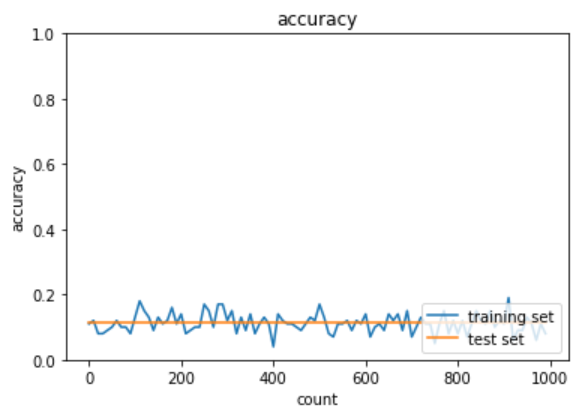
・RMSPropの過去勾配の2乗の指数関数的減衰平均

* **実装演習結果(キャプチャーorサマリと考察)**

〇2\_4\_optimizer\_after.ipynb

・SGD

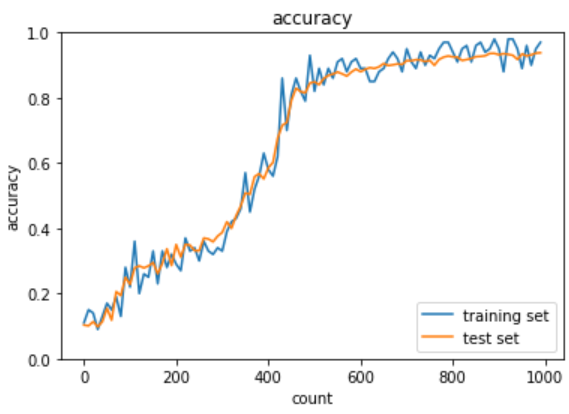
学習がうまくいっていないことを確認(1000回の学習で精度11%程度)



・Momentum

学習がうまくいくことを確認(1000回の学習で精度93%程度)

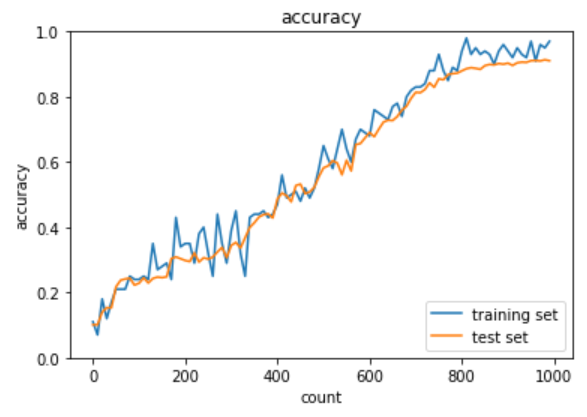
※動画ではうまくいっていない(よい初期値が割り当てられた結果と考える)



・AdaGrad

学習がうまくいくことを確認(1000回の学習で精度91%程度)

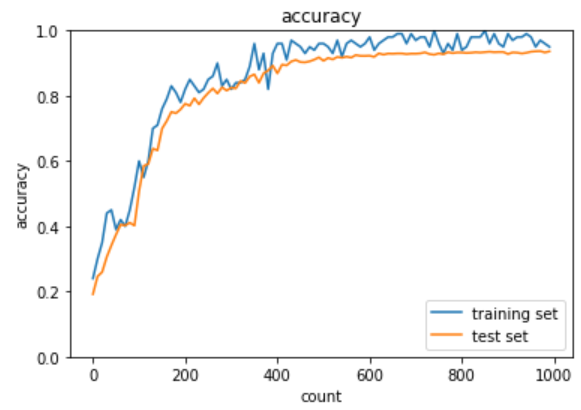
学習効率はMomentumより低い。



・RMSprop

学習がうまくいくことを確認(1000回の学習で精度93%程度)

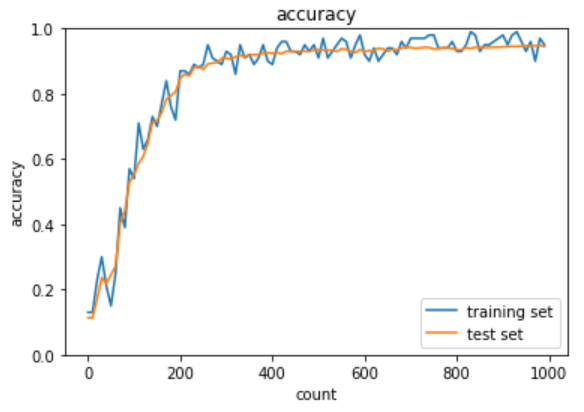
学習効率はMomentumより高い。



・Adam

学習がうまくいくことを確認(1000回の学習で精度94%程度)

学習効率はRMSpropと同等であった。



アルゴリズムの経緯、及び確認結果からパラメータ更新方法としてAdamを採用すればよいと確認できた。

* **確認テストなど、自身の考察結果**

〇モメンタム、AdaGrad、RMSPropの特徴

・モメンタム

SGDに 移動平均を適用して、振動を抑制する。大域的最適解になりやすく、最適化までの収束時間が早いメリットがある。

・AdaGrad

過去の更新量をもとに徐々に学習率を下げることで振動を抑制する。勾配が緩やかな斜面に対して最適解に近づけるが、学習率が徐々に小さくなるので鞍点(あんてん)問題が発生する場合がある。

・RMSProp

AdaGradの問題を解消した手法であり、大域最適解となりやすく、まハイパーパラメータθの調整が容易である。

* **演習問題、参考図書、修了課題などの関連レポート**

最適化手法の動きについて、以下のサイトで再確認した。

https://qiita.com/omiita/items/1735c1d048fe5f611f80

github.com/Jaewan-Yun/optimizer-visualization

* 1. **Section3：過学習**
* **要約**

過学習とは訓練誤差とテスト誤差で学習率が乖離することであり、特定のサンプルに特化し過ぎた状態。傾向として、データ量が少なくNNのノード数が多い場合に発生しやすい。過学習の対策として、以下2点が挙げられる。

①正則化(L1正則化、L2正則化)

②ドロップアウト

①について、過学習の原因と対策は以下の通り。

・過学習の原因

学習させていくと、重みにばらつきが発生する。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きる。

・過学習の対策

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、かつ重みのの大きさにはばらつきを出す必要がある。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

②について、過学習の原因と対策は以下の通り。

・過学習の原因

ノード数が多い

・過学習の対応

ランダムにノードを削除して学習させること

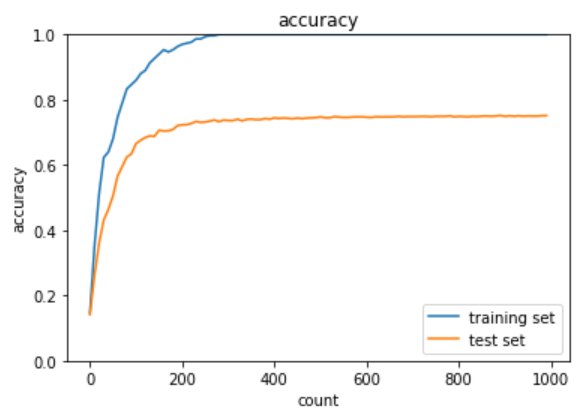
データ量を変化させずに、異なるモデルを学習させていると解釈できる。

* **実装演習結果(キャプチャーorサマリと考察)**

〇2\_5\_overfiting.ipynb

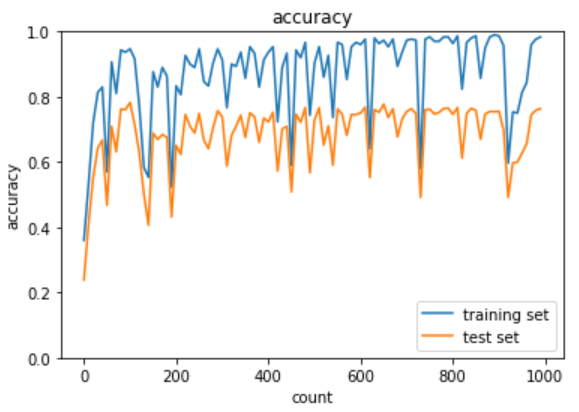
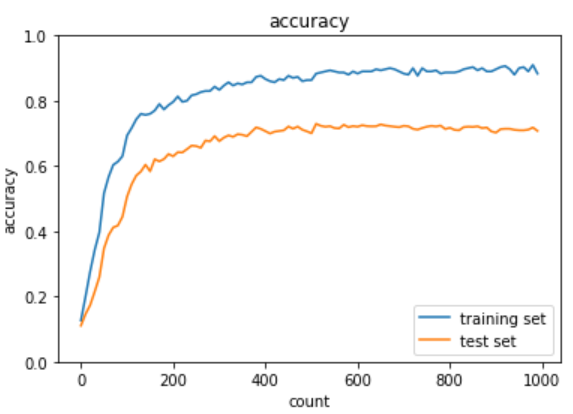
・overfiting

6個の中間層にそれぞれ100ノードを保持するNNに対し、300件から100件を抽出し、1000回学習を実施。結果、過学習となっていることを確認。



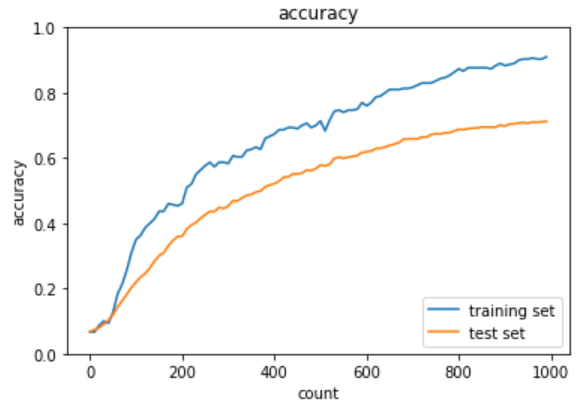
・L2正則化、L1正則化

過学習ではなくなった(訓練データでの精度が100%から80%程度に下がった)が、今回の例では検証用データでの精度は上がらなかった。



・ドロップアウト

学習率の曲線が理想に近い形である。また、未学習の状態であり、良い結果が得られそうである。



・ドロップアウト＋L1正則化

上記のドロップアウトでは未学習状態であったが、当ケースでは収束に近づいているように見え、学習効率が上がっていると考える。

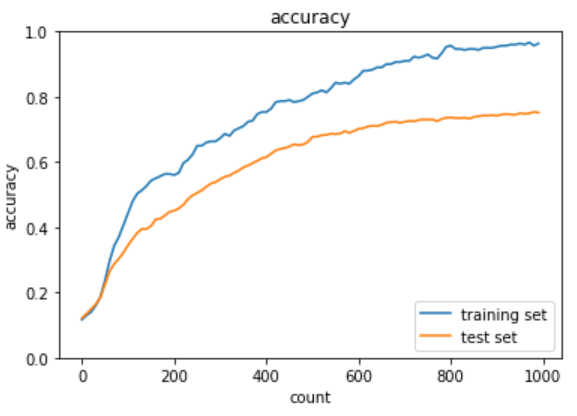
* **確認テストなど、自身の考察結果**

〇L1正則化を表している図はどちらか

x軸、y軸は重み、等高線は誤差関数の等高線を表しており、L1正則化は重みを繋ぐ線がひし形となる。

〇例題チャレンジ

L1ノルムの微分がsin関数となることを確認した。



* 1. **Section4：畳み込みニューラルネットワークの概念**
* **要約**

CNNは次元間で繋がりのあるデータを扱える。

CNNは入力層、畳み込み層(convolutions)、プーリング層、全結合層、出力層で構成される。LeNetの構造図を確認。

「入力層＋畳み込み層＋プーリング層」は次元の繋がりを保ち、特徴量抽出を行っている。「全結合層＋出力層」は上記の特徴量から取得したい形式に変換している。

〇畳み込み層

・全体像

入力値(3チャンネル4×4)にフィルタ(3チャンネル3×3)を通すことで、3チャンネル2×2の結果が得られる。上記にバイアスを足し、活性化関数を通して出力値を得る。

フィルタは周囲の情報を集約するため、データの繋がりをうまく表現することができる。

画像の場合、縦、横、チャンネルの3次元データをそのまま学習し、次に伝えることができる。

・パディング

畳み込みにより次元が小さくなり不都合(次元が消失する等)が発生する。

その対応としてパディングがあり、これにより畳み込みによる次元の減少を回避できる。

パディングは、入力データの次元を広げ、固定値(0や近傍の値を設定するなど)を設定する。拡張された入力データに対し、畳み込みを行う。

・ストライド

ストライドは、入力データに対しどのようにフィルタを当てるか(1コマずつずらす、等)を設定する。これにより、出力の次元をコントロールできる。

・チャンネル

チャンネル数はフィルタの数である。個性を持ったフィルタを用いることで表現力を上げることができる。

〇プーリング総

対象領域の最大値や平均値を取得する。

* **実装演習結果(キャプチャーorサマリと考察)**
* **確認テストなど、自身の考察結果**

〇6×6の入力画像を2×2のフィルタで畳み込みを行った場合の出力サイズ(ストライド、パディングは1)

出力サイズを計算する公式を確認した。

* **演習問題、参考図書、修了課題などの関連レポート**

〇2\_6\_simple\_convolution\_network.ipynb

・image to column

計算高速化のためのim2col関数の挙動を確認した。np.pad、transposeが難解だった。

・col2im

ベクトルから元の次元へ変換する方式を確認した。

・convolution class

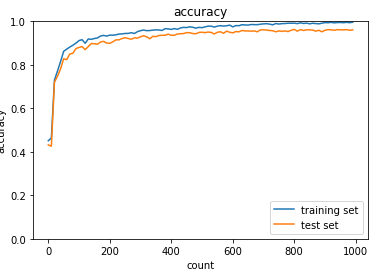
順伝播、逆伝播共に行列の形式の変換後、線形演算を行い、行列を元の形式に変換している。

・pooling class

行列の形式変換後、maxプーリングを行い、元の行列の形式に戻している。

・sinple convolution network class

畳み込み層、ReLU、プーリング、全結合、ReLU、全結合、softmaxのネットワーク構造。畳み込み層は、フィルタ数30、フィルタサイズ5、パディング0、ストライド1。学習にはAdamを利用。非常に高い精度が得られた。



* 1. **Section5：最新のCNN**
* **要約**

AlexNet

畳み込み層、及びプーリング層を含む5層、それに続く3層の全結合層から構成される。過学習を防ぐため、サイズ4096の全結合総の出力にドロップアウトを使用している。

全結合層の計算方法として、以下が挙げられる。

　・Fratten：13×13の256チャンネルをベクトルに変換(43264個)

　・Global MaxPooling：チャンネル毎の最大値を取得する(256個)

　・Global AveragePooling：チャンネル毎の平均値を取得する(256個)

Frattenより、Global MaxPoolingやGlobal AveragePoolingの方が機能することが多い。

* **演習問題、参考図書、修了課題などの関連レポート**

AlexNetについて、以下のサイトで追加学習を行った。

https://axa.biopapyrus.jp/deep-learning/cnn/image-classification/alexnet.html

2012 年に、人が特徴量を設計しなくても、十分なデータさえ存在すれば、機械自身が特徴量を見つけ出すことが AlexNet によって示された。