【識別器の開発アプローチ】

機械学習の識別機には、大きく生成モデルと識別モデルと識別関数の3種類のアプローチがあるある。

・生成モデル

画像を作り出すモデルであり、あるクラスの条件の下で画像ができる。

・識別モデル

ある画像の下で、どのクラスに属するかを判断するモデルである。クラスに属するかの確率を出力とする。確率に基づいて識別が可能である。

・識別関数

データの属するクラスの情報のみを出す。決定的な識別を行い、学習量が少ないのが特徴である。

生成モデルと識別モデルの計算はベイズの定理より関係がある

【識別機における生成モデルと識別モデル】

生成モデルはデータの複雑さをモデル化する。データのクラス条件付き確率を求める。確率分布を推定するため計算量が多い。これに対し、識別モデルはデータの中身はあまりモデル化しない単に分類結果を得るのみなので、直接データがクラスに属する確率を求める。

【識別モデルと識別関数】

識別モデルは識別結果の確率が得られる。この推論結果に基づいて識別結果を決める必要がある。識別関数は識別結果のみをいきなり決める。識別モデルは確率を出すため、判断の確実性を評価できるので推定結果から判断を変更可能。

【万能近似定理と深さ】

ニューラルネットワークはどんな関数でもよく近似できるという考えである。

【NN（ニューラルネットワーク）全体像】

ニューラルネットワークは入力層・中間層・出力層の3つから構成される。

・入力層

入力は数字の集まりであり入力層はその入力情報を受け取る部分である。

・中間層

学習・推定処理する部分である。

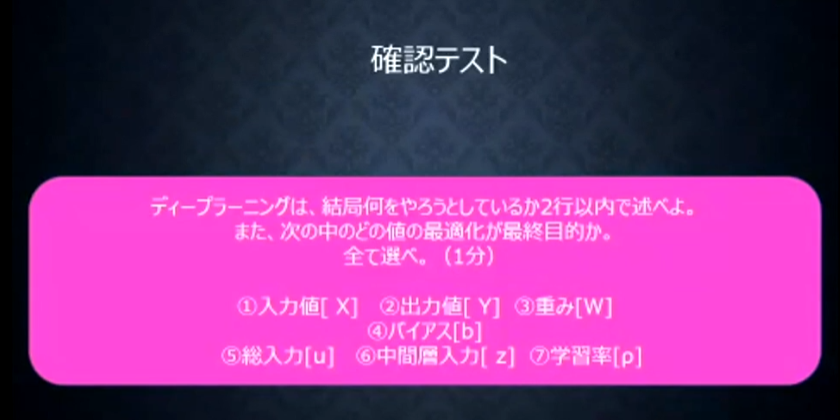
・出力層

最終結果を出力する部分である。

ニューラルネットワークでは

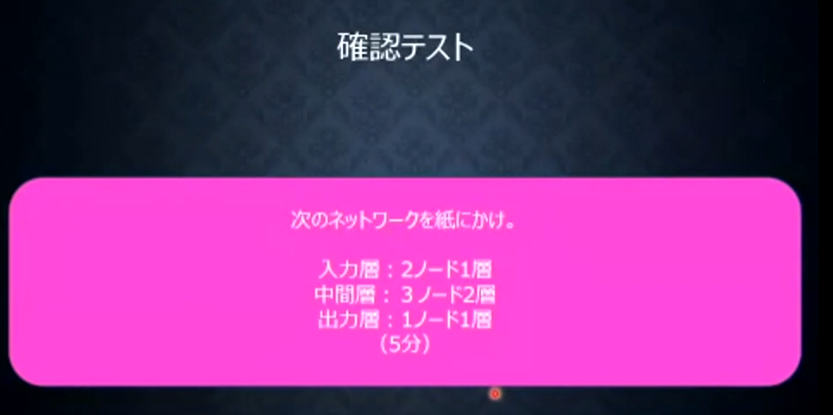
W:重み　B：バイアス　というパラメータで入力信号を変換していく。

確認テスト

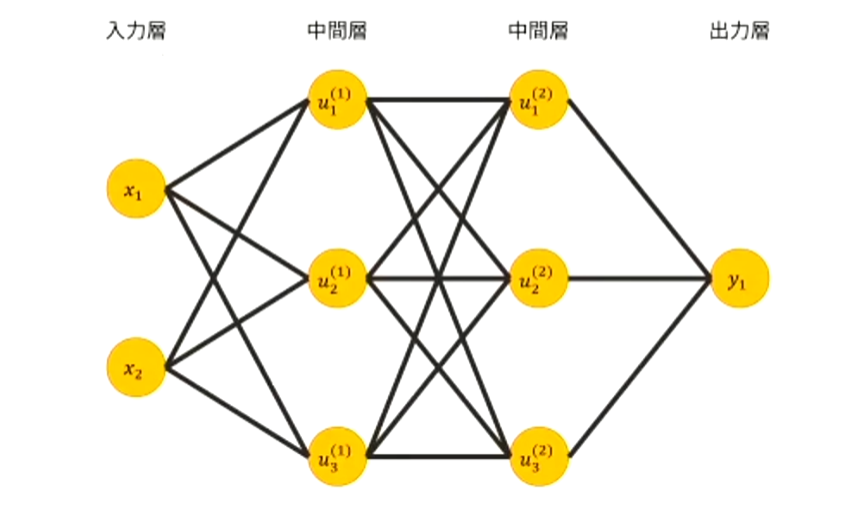


ディープラーニングは何をしようとしているかというと、入力値から目的とする出力新井に変換する数学モデルを自力で構築しようとしている。最終的には重み、バイアスが調整されて得られる。

確認テスト



題意を満たすネットワークは以下のようになる。



ニューラルネットワークは変換器であり、入力に対して変換して出力を得る。用意した正解とその出力を比較して正しいかどうかをニューラルネットワークにフィードバックする。そのフィードバック結果を受け、重みとバイアスを調整する。これを繰り返すことが学習であり、学習を行うことで、推定精度の高いニューラルネットワークが完成する。

ニューラルネットワークが対象とする問題は回帰と分類問題がある。

回帰問題は例えば、連続データの予想や売り上げ予想や株価予想、ランキング問題などが該当し、連続する実数地をとる関数の近似を行うことが目的である。使われる機械学習モデルは線形回帰やランダムフォレスト、ニューラルネットワークなどがある。

分類問題は例えば、性別や動物の種類など離散的な結果を予測するための分析が該当する。写真の判別、何の画像かを分類と行うことが目的である。使われる機械学習モデルはベイズ分類、ロジステッィク回帰、NNなどがある。

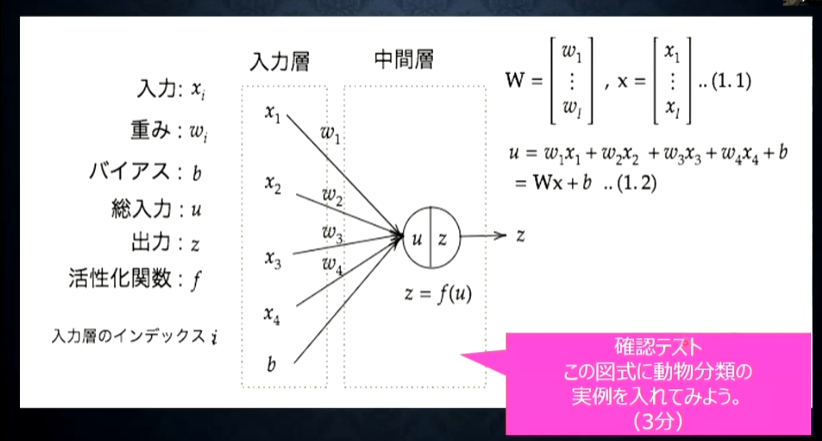
ニューラルネットワークは回帰、分類両方に使えるモデルであり、万能近似定理を支持する根拠の一つである。

深い層のニューラルネットワークは深層学習モデルと呼ばれる。深層学習モデルは、より複雑な問題を解くことが可能であり、自動売買やチャットボット、翻訳などに実装されるモデルが該当する。数字の集まりを入力にとり、数字の集まりの出力を得るものなら基本的になんでも適用可能である。これがニューラルネットワークの強みである。もともと数字でなくても、文字や画像も数字化し、数字で表現して置き換えることで、ニューラルネットワークで扱える問題に変換することで適用できるようになる。音声も数値変換可能であるため音声解釈が可能である。ニューラルネットワークは囲碁、将棋AIにも適用されている。

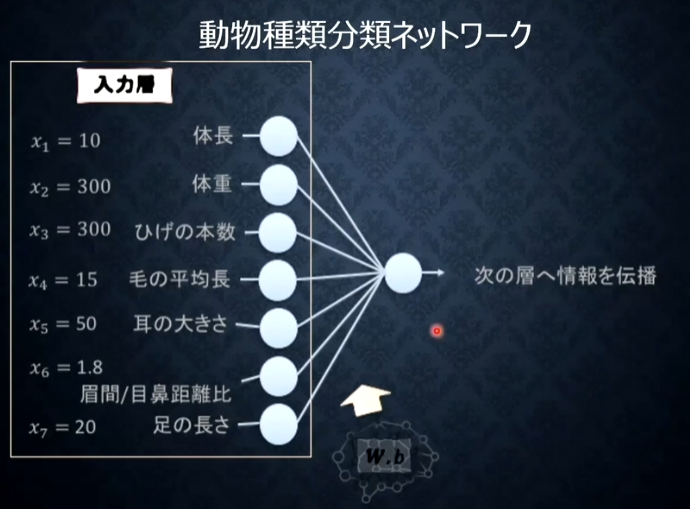
【入力層から中間層】

入力を受け取る場所はノードと呼ばれる。その値入力をどれくらい使うかを決めるのが重みＷであり、入力の値を重みで使用割合程度を決めて組み合わせる。重みも入力も行列であらわされ、それら内積で出力が得られる。PythonではNumpyのdot関数で計算可能である。バイアスＢは直線式の切片のようなもので、すべての入力に対して一定の変化を与える。重みは直線式において傾きに対応する。傾きと切片が決まれば直線が決まる、すなわち出力が決まる。Ｗ、Ｂによって計算された出力は層入力と呼ばれる。中間層において層入力に活性化関数を適用し、そしてまた次の入力に引き渡される。入力と中間層が1つのセットでありこれが何個も繰り返される。

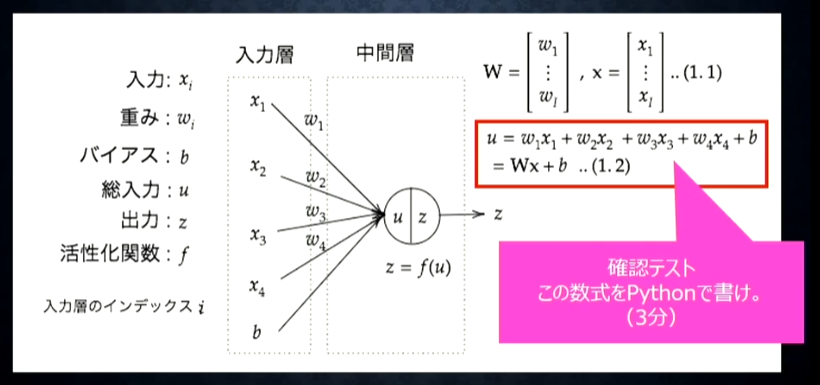
確認テスト



回答は以下のようになる。体調や体重など、動物の分類に必要な特徴（説明変数）を入力層に用意する。



サンプルコード演習+確認テスト

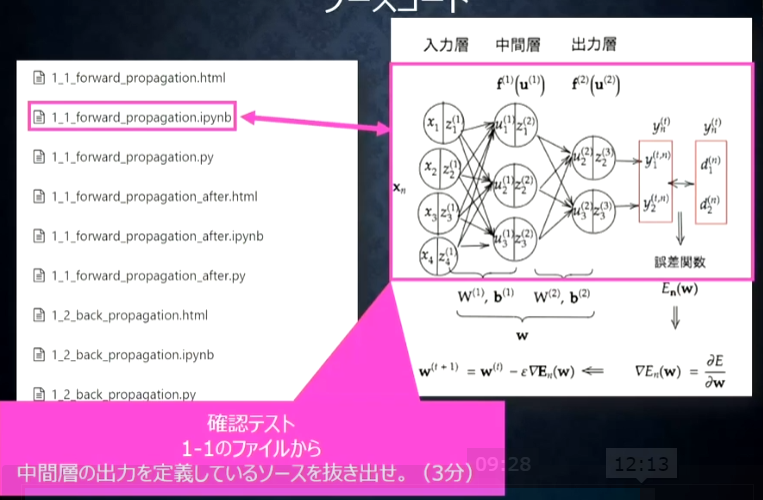


順伝播・単層・単ユニットというサンプルコードにおいて、

U=np.dot(x,W)+b

というコードが、上記確認テストの回答となる。Numpyというライブラリを使用しており、array()で行列を生成し、np.dot()で行列の内積を計算可能である。

サンプルコード演習+確認テスト



順伝播（3層・複数ユニット）のサンプルコードにおいて、

u2=np.dot(z1,W2)+b2

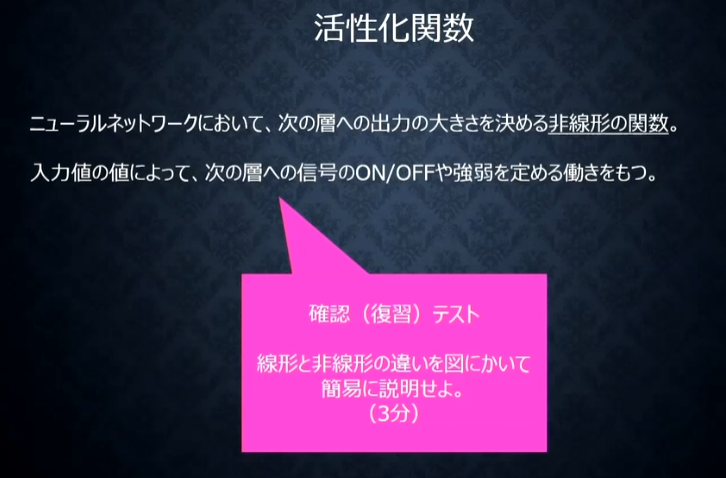
z2=functions.relu(u2)

の部分が中間層の出力を定義しているソースとなる。

【活性化関数】

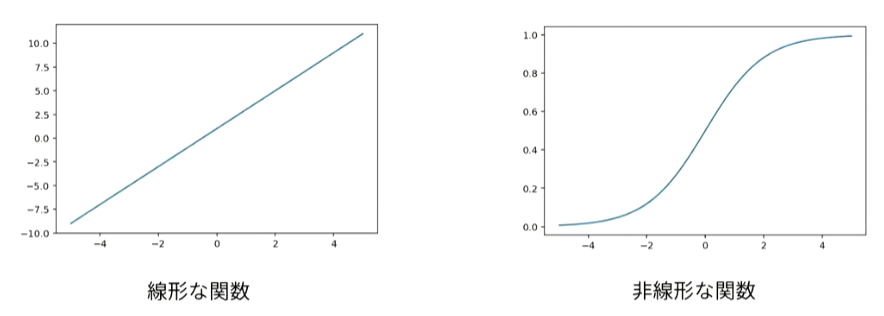
ニューラルネットワークで次の層への出力の大きさを決める非線形の関数が活性化関数と呼ばれるものである。入力値の値によって、次の層への信号のON/OFFや強弱を決める働きを持つ。

確認テスト



回答

線形・非線形関数は図に表すと下図のようになる。線形とは、加法性（2つ引数の和の関数出力が、それぞれの引数に独立に関数を適用した値の和と等しい）と、斉次性（関数の引数の実数倍は元の引数の関数出力値に後から実数倍しても等しくなる）の2つの性質を満たす。一方、非線形関数は、加法性、斉次性を満たさない関数である。



実際に活性化関数は重みとバイアスによる入力値の計算がされた後に中間層で適用される。重みとバイアスによる入力値の計算は線形計算だが、活性化関数は非線形計算であり、NNのモデル表現の幅を広げるカギとなっている。活性化関数にはいくつか種類がある。中間層に主に利用され活性化関数は

・ReLU関数

勾配消失問題を解消可能であり、ニューラルネットワークをスパース化することが出来る関数である。

・シグモイド関数

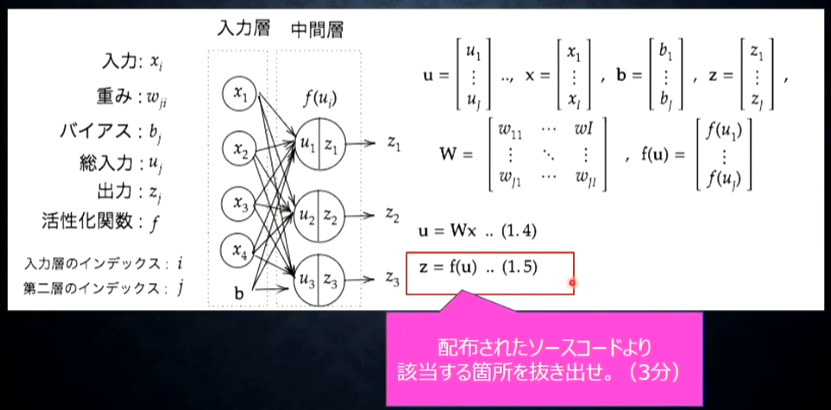
非線形関数で、数式ではｆ＝1/（1+ＥＸＰ（-u））と表す。ステップ関数を緩やかに変化させたような関数である。微分可能であるため、学習を独立に進めることが可能である。複雑なニューラルネットワークでは勾配消失問題が発生してしまう場合がある。

・ステップ関数

現在ではあまり使用されない関数であり、閾値を超えたら瞬時に1となる0か1の出力しかとらない関数である。

などがある。出力層で主に使用される活性化関数はソフトマックス関数、恒等写像、シグモイド関数などがある。

サンプルコード演習+確認テスト



順伝播（3層・複層ユニット）のサンプルコードにおいて、活性化関数を使っている部分は、

z1=functions.relu(u1)

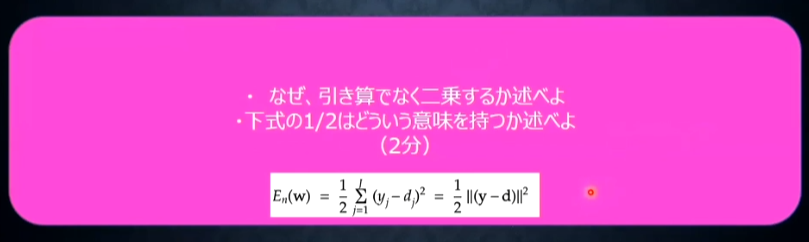
である。別のファイル（functions.py）にrelu関数が定義されている。

【出力層\_誤差関数】

出力層はニューラルネットワークで一番最後の層である。出力層は各クラスごとの確率を出力するようにする必要がある。

ニューラルネットワーク全体でどのように学習させるかについて説明する。まず訓練データを用意する。入力データと、訓練データ（正解値）の1対1のぺアをたくさん用意する。実際に学習させるとき入力データをニューラルネットワークのインプットとする。ニューラルネットワークの出力で各クラスの確率が出力される。出力結果と訓練データとを比べ、あっているか間違っているかを評価する。このときどのくらいあっていたかを表現するために誤差関数を使う。誤差関数には二乗和誤差などがある。誤差関数の数値が小さいほど誤差が小さいことを意味する。二乗和誤差はそれぞれのクラスの出力結果と正解値を引き算し、二乗したものの合計を計算する。

確認テスト



2乗しないと引き算した結果に正負両方発生し、全体の誤差を正しく表せない。このため2乗してそれぞれのラベルでの誤差を正の値になるようにする。1/2の定数をかける理由は、微分した際に表式が簡単になるからである。

プログラムソース確認

「回帰」ニューラルネットワークサンプルプログラムの中で、Functions.mean\_sqared\_error()というコードで平均二乗誤差の誤差関数が使われている。分類問題ではクロスエントロピー誤差を用い、回帰問題では平均二乗誤差を用いることが多い。

【出力層の活性化関数】

中間層の活性化関数は閾値の前後で信号の強弱を調整し、次の入力層で必要な信号を抽出する役割を持つ。一方、出力層の活性化関数は信号の比率はそのまま保ち変換する。最終的に得たい出力の形に加工する役割を持つ。よって出力層と中間層の活性化関数では目的・役割が異なる。確率出力に関して、分類問題では、出力層の出力は0～1の範囲に限定し、総和が1となるようにする必要がある。出力層で用いられる活性化関数は大きく分けて以下の3つがある。

・恒等関数

回帰問題に使い、誤差関数は二乗誤差を採用する。

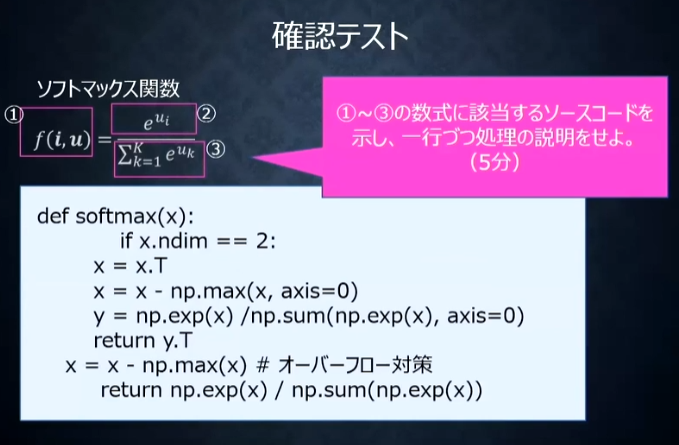
・シグモイド関数

二値分類に使い、誤差関数は交差エントロピーを採用する。シグモイド関数は0～1の間をなだらかに変化する関数である。

・ソフトマックス関数

他クラス分類に使い、誤差関数は交差エントロピーを採用する。全体の総和をとると1になり、これを満たすように式の分母に総和の項がある。

サンプルコード演習+確認テスト



Commonフォルダのfunctions.pyのソースプログラムにおいて、

1. softmax(x)：ソフトマックス関数を定義し出力を返す。
2. np.exp(x)：各要素の指数関数出力を計算する。
3. np.sum(np.exp(x))：各要素の指数関数出力の総和を計算する。

コード確認

二乗平均誤差のコードは

def mean\_squared\_error(d,y):

return np.mean(np.square(d-y))/2

となる。

サンプルコード演習+確認テスト



Commonフォルダのfunctions.pyのソースプログラムにおいて、

1. Cross\_entropy\_error(d,y)：交差エントロピー関数を定義し出力を返す。
2. -np.sum(np.log(y[np.arange(batch\_size),d]+1e-7))/batch\_size：各要素の対数をとり、合計してマイナス符号をつける計算が行われている。+1e-7は対数関数の出力が発散しないように加えられている。

【勾配降下法】

深層学習の目的は学習を通して誤差を最小にするパラメータ（重みW,バイアスB）を探索することである。このとき勾配降下法を利用してパラメータ探索が可能となる。

前回の学習結果の重みを更新して、次の学習の重みとする。これを繰り返す。

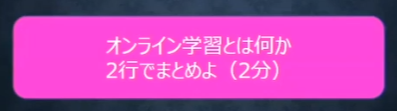
学習率とは、どれだけ更新するかの尺度を表すパラメータである。これをうまく設定することで、学習効率が良くなる。学習率が大きすぎると永遠に学習が完了せず、パラメータが最適解を見つけられず発散する。かといって、学習率が小さいと学習にかかる時間が増大する。また、ローカルな極小値で学習が終わってしまい真に得たいより小さい値を探索できない。

勾配降下法では、Momentumや、Adadelta、AdaGrad、Adamなど様々な手法が提案されている。入力して出力された結果と訓練データから学習時は誤差関数の結果から、ｗ，Ｂを更新する。このサイクルが学習であり、この1サイクルをエポックと呼ぶ。通常このエポックを何百回と繰り返し精度の良いニューラルネットワークモデルを得ることが出来る。

【確率的勾配降下法ＳＧＤ】

確率的勾配降下法では学習する全データから、ランダムに1部分だけ選びだし学習をする。学習するデータを少なくすることで計算コストを軽減できる。また、学習ことに使用するデータが異なるので、望まない極小解に収束するリスクを避けることが出来る。また、オンライン学習できることもメリットである。オンライン学習は、最初にすべてのデータを準備する必要がなく、リアルタイムに徐々に得られるデータに対して、都度逐次的に学習が可能となる。一方バッチ学習では一度にすべての学習データを使ってパラメータ更新を行う。バッチ学習は1度に大量の学習データが必要なものから、ＰＣのメモリに記憶できるデータの兼ね合いから実行できない場合がある。この場合はオンライン学習が有効である。

確認テスト

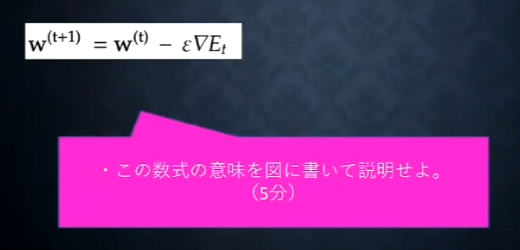


オンライン学習とは、学習データが入ってくるたびに都度パラメータを更新し、学習を進めていく方法である。

【ミニバッチ勾配降下法】

オンライン学習の特徴をバッチ学習に応用した勾配降下法がミニバッチ勾配降下法である。ミニバッチ勾配降下法ではまず、学習データを小分けにする。これをミニバッチと呼び、分割した個数で勾配計算結果を割ることでミニバッチあたりの更新量を求めていく。ミニバッチ勾配降下法では、確率的勾配降下法のメリットを損なわずに、計算機の計算資源を有効利用できる。例えば、ＣＰＵを利用したスレッド並列化やＧＰＵを利用したＳＩＭＤ並列化と組み合わせて適用することが出来る。すなわち、ミニバッチに分けることで、同じ学習を同時並行的に並列処理が可能となる（SIMD:Single Instruction Multi Data）。

確認テスト



回答図は以下のようになる。同じ問題に対し何度も学習していくことに相当する。学習でどれだけ間違えたか▽Eｔに学習率εをかけて、次の時刻のエポックにおいて、重みＷを更新していく。



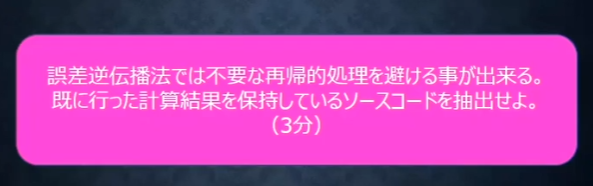
【誤差勾配の計算】

重みはニューラルネットワークではたくさん登場する。誤差の変化に合わせ各々の重みをどう変えていくかを数値的に得るために微分を使う。微分の方法として、数値微分はプロクラムで微小な数値を生成し疑似的な微分の計算をする一般的な方法がある。しかし、数値微分は計算負荷が非常に大きいことがデメリットである。ニューラルネットワークの層が増えていくと、微小増加量、微小減少量を重みごとに計算しなくてはならないため、その計算も増加する。

【誤差逆伝播法】

数値微分では、計算コストがかかっていたが、誤差逆伝播法その問題を解決した計算方法である。。算出された誤差を、出力層から逆順で微分し、前の層へと伝播させていく。このように誤差逆伝播法は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となっている。誤差から微分を逆算し、不要な繰り返し計算を避けて微分を算出できる。これは微分の連鎖律を利用している。一回計算した内容を、さかのぼりながら再利用できる。各層の微分をそれぞれ計算し、掛け合わせる。

サンプルコード演習+確認テスト



1\_3のコードの確率勾配降下法の部分で、

delta2=functions.d\_mean\_squared\_error(d,y)

delta1=np.dot(delta2,W2.T)\*functions.d\_sigmoid(z1)

が該当するソースコードである。

サンプルコード演習

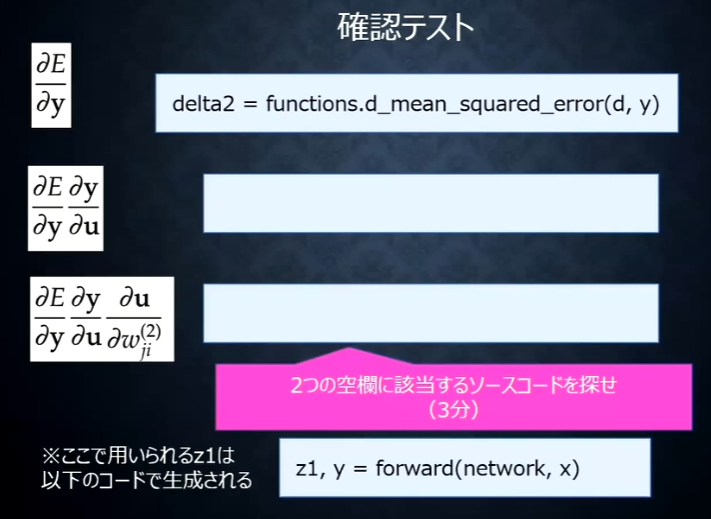
1\_3のコードの確率勾配降下法の部分で、

出力層ではＥをｙで微分した結果がdelta2 ＝functions.d\_mean\_squared\_error(d,y)であらわされている。シグモイド関数の微分形が掛け合わされて、

delta1=np.dot(delta2,W2.T)\*functions.d\_sigmoid(z1)

と次の層の誤差が計算されている。

サンプルコード演習+確認テスト



1\_3のコードの確率勾配降下法の部分で、

ΔＥ/Δｙに該当するコード：delta2=functions.d\_mean\_squared\_error(d,y)

ΔＥ/Δｙ・Δｙ/Δuに該当するコード：delta1=np.dot(delta, W2.t)\*funcions.d\_sigmoid(z1)

ΔＥ/Δｙ・Δｙ/Δu・Δu/Δwに該当するコード：grad[‘W1’]=np.dot(x.T,delta1)

となる。

【ディープラーニングの開発環境】

ローカルでは、CPU（どんなパソコンにも入っており、安く汎用性がある計算機。ＧＰＵ等に比べ計算速度が遅い）やGPU（ゲーム用のPCに入っている計算機）、FPGA（自分でプログラミングが実装でき、かなり高価な計算機。）、ASIC（CTPが例としてあり、自分でプログラミングできない計算機。汎用性がなく非常に高価だが、最も計算が速いのが特徴）といった計算機が必要である。TPUはGoogleが機械学習のためだけに作ったASIC計算機であり、クラウドサービスも存在する。ローカルとは家にあるパソコンが該当する。一方クラウドではAWSやCCPが必要であり、クラウドはデータセンターを意味する。

【入力層の設計】

入力として取りうるデータは、連続する実数や確率、01のフラグ値（one-hotラベル）、のような数字の集まりである。一方、入力層として使えないデータは、いかに上げるようなデータが該当する

・欠陥値が多いデータ

→欠損値は学習には役に立たないデータ。

・誤差の大きいデータ

・出力そのもの。

・出力を加工した情報

→基本的には人下の手が介入しない生データを入力とするべきであるが、何かの変形を経て学習させてしまうと、その変形を受けてよい学習ができない。あるニューラルネットワークの出力を別のニューラルネットワークの入力にしないということも気を付けておくべきである。

・連続性のないデータ

→背番号とか、単に番号が振られたデータ

・無意味な数が割り当てられているデータ。

→違う意味なのに同じ数字が割り当てられている等

欠損地の扱いは主に以下の3通りある。

・ゼロで埋める

・欠損地を含む集合を除外する

・入力として採用しない

数値の正規化（0～1に収める）や、正則化（正規分布に従わせる）もデータの統合で行われる。

【過学習】

学習データにモデルが過剰に適合してしまい、汎化性能が失われている現象のことを過学習という。過学習は巨大なニューラルネットワークで発生することが多い。過学習を見分けるには、検証用のデータで予測をしてみると予測精度が悪くなるという傾向から判断する。過学習を防ぐ方法としてドロップアウトがある。

【データ集合の拡張】

データセットが少ない場合の増やす手法（Data Augmentation）について説明する。データが不足するとき人工的にデータを増やす場合がある。これは分類タスクのモデルで効果が高い。一方、連続データの分布予測では効果がない。画像でのデータ拡張方法は明るさを変えるオフセットやノイズを入れたり、一部を隠すドロップアウトなどがある。これら様々な変換を組み合わせて水増しデータを生成していく。データ拡張の時、データセット内で混同するデータが発生しないように注意する。例えば6と9の画像の回転などが混同例に該当する。

ニューラルネットワーク内にノイズを入れることでデータを拡張する方法もある。中間層が深くなるごとに、特徴が抽象化・複雑化され、抽出される。それぞれの中間層で、それぞれの特徴レベルに応じてデータ拡張することが可能である。データ拡張の方がモデル改良余地も、汎化性能を劇的に伸ばす場合がある。しかし、データ拡張を行うと、何個かのモデルの実験比較ができなくなるという問題がある。この場合はデータ拡張方法を統一して、同じデータを使うようにする。このため、データ拡張はモデルの1部としてとらえられることがある。

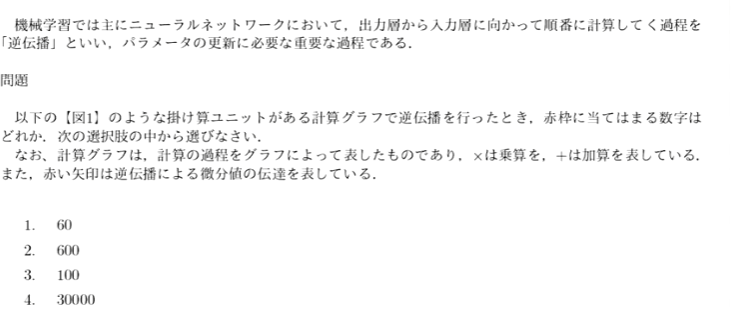
【CNNで扱えるデータの種類】

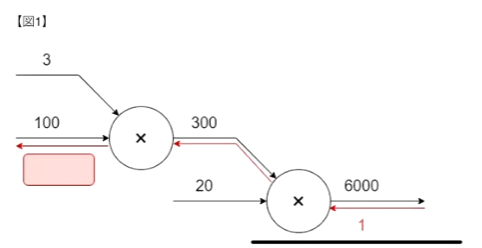
CNNでは、次元間でつながりがあるデータを扱える。例えば単一チャンネルでは、音声や、フーリエ変換した音声等がある。複数チャンネルでは、カラー画像や動画が該当し、これらのデータの変化には連続性がある。すなわち急に非連続な変化をしないデータがＣＮＮで扱えるデータとなる。次元数とチャンネル数によってデータの種類が分類されるが、そのすべての種類のデータにCNNは適用可能である。

【特徴量の転移】

転移学習という学習手法に関して説明する。深層学習モデルにおいて、中間層（入力層に近い側）では画像内の各パーツの基本的な特徴が抽出されている。この基本的な特徴量抽出を別の大量なデータで学習しておく（プリトレーニング）。この学習を前もって行っておき、基本的な特徴量抽出しておいて、他の深層学習モデルにパラメータ（重み）を流用する。プリトレーニングは学習なしでトレーニングする。画像ではVGG、自然言語処理ではVERTなどが、プリトレーニングとして有名なモデルである。学習のコストを抑えつつ高精度なモデルが出来る。これを転移学習と呼ぶ。ファインチューニングでは学習済みの重みも再学習する。

【演習問題】



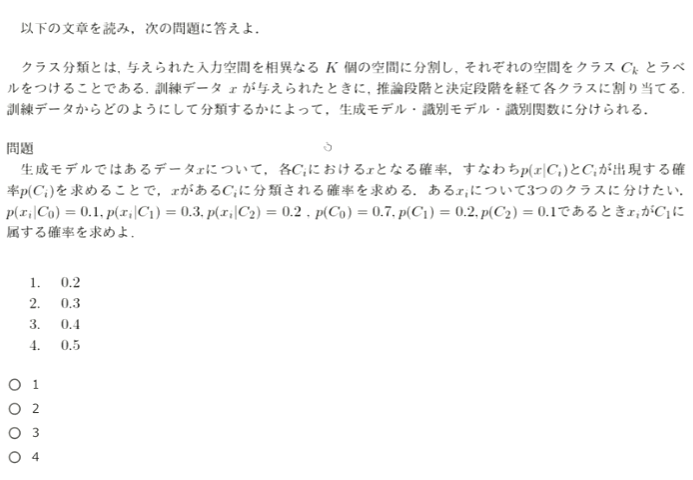


積の層では、逆の枝の乳勅をかけることで、逆伝播を計算できる。よって

赤枠に該当する数値は、

1×20×3＝60

となる

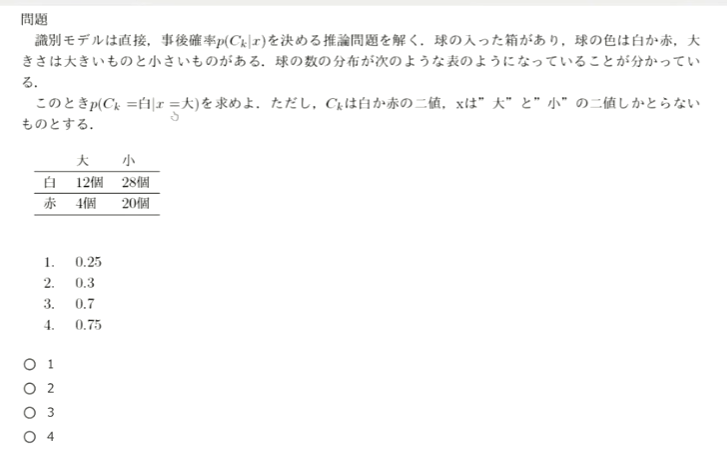


事後確率はベイズの定理を用いるとp(C1|x)=p(x|C1)p(C1)/p(x)である。

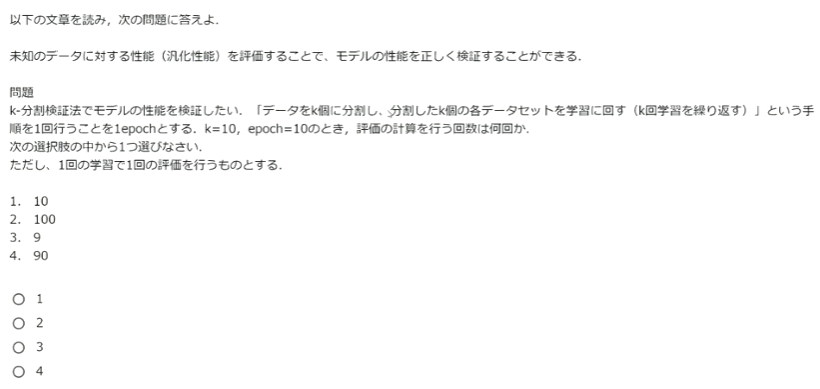
ばた、p(x)=∑p(x|Ck)p(Ck)であるから、問題から与えられた確率の条件より,

求める確率は0.3\*0.2/(0.1\*0.7+0.3\*0.2+0.2\*0.1)=0.4

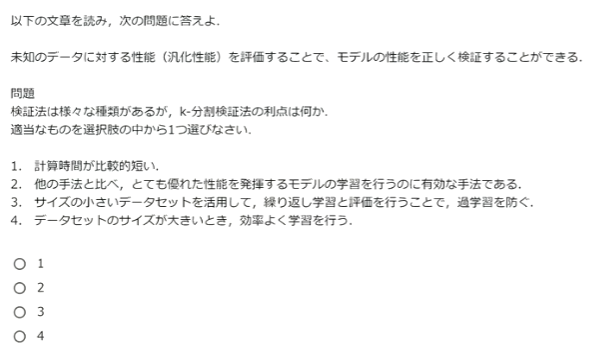
となる。



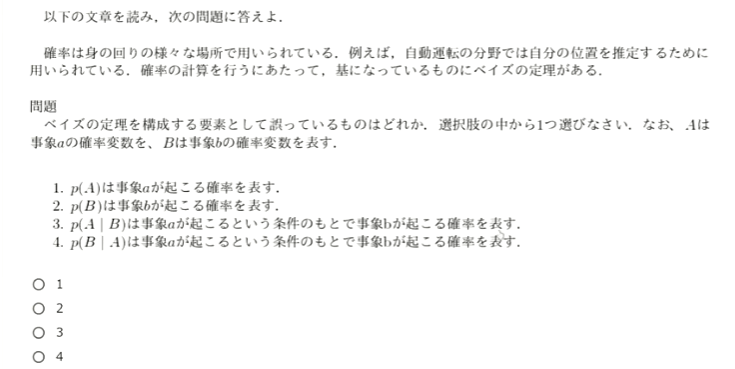
P(Ck=白|x=大)は球の大きさが大きいときに色が白である確率を表す。大きい球は、白と赤合わせて16個あるため、求める確率は12/16=0.75である。



データを10個分割し、それを10エポック学習させるので、実際の評価の計算を行う回数は10×10＝100回である。



K分割検証法において、例えばモデルを5分割した場合、5回学習と検証を繰り返す。このようにサイズの小さいデータセットの場合でも、学習と検証回数を増やし精度を高めることが出来る手法である。よって正解は3となる。なお、繰り返し学習と評価を行うため、計算時間がかかるという欠点がある



P(A|B)は条件付き確率を表し、事象bが起こるという条件の下での事象aが起こる確率を表すため、3の選択肢が誤っている。

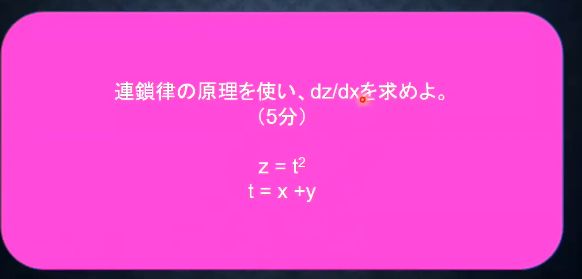
【深層学習の復習】

深層学習では、入力層に値を入力し、重み、バイアス、活性化関数で計算しながら伝わった後、出力層から値が得られる。出力層の値と正解値から誤差関数を使って誤差を算出する。誤差の結果をモデルにフィードバックする。微分の連鎖律を使った逆伝播で誤差を小さくするために重みやバイアスを更新する。これらを何回も繰り返し。重み、バイアスをどんどん更新していく。これにより出力値を正解値一に近づける。

複雑なデータを処理するために中間層を増やしていくことが必要とされる。例えばAlex netでは5層の畳み込み層とブーリング層、そして3層の全体結合層から構成される。畳み込みニューラルネットワークは画像認識に使われるが、画像を扱う場合、入力値が多くなる。そのような複雑な入力の場合は中間層を増やす必要がある。

【誤差逆伝播法・勾配消失問題】

確認テスト



dz/dt=2t ,dx/dt=1より、dz/dx=dz/dt・dt/dx=2t・1=2t

サンプルコード演習

1\_3\_stochastic\_gradient\_decent.ipynbのソースコードの確率的逆誤差伝播法のプログラムにおいて、

def backward：

という部分で逆誤差伝播法の計算が実装されている

delta2=functions.d\_squared\_error(d,y)

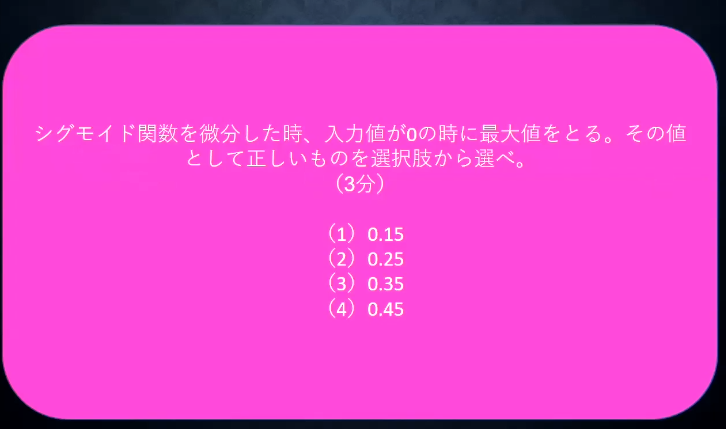
という部分がｄをｙについて微分した関数となっている。

また、delta1=np.dot(delta2,W2.T)\*functions.d\_sigmoid(z1)

という部分が、シグモイド関数の微分積が登場し、誤差逆伝播の微分連鎖律の計算が行われている。

深層学習では中間層を増やす必要があることを前述したが、中間層を増やしすぎると勾配消失問題が発生する。誤差逆伝播をするとき微分の連鎖律を利用するが、中間層を増やしていくと、微分の項がどんどん増えていき、勾配がどんどん緩やかになる。そのため、勾配逆伝播による更新では、階層のパラメータはほとんど変わらず、学習しても最適解に収束しなくなる。これを勾配消失問題と呼ぶ。微分値は0～1の範囲の値となる場合が多く、0～1の値を何回もかけると値がどんどん小さくなる。シグモイド関数は勾配消失問題を起こす代表的な関数である。シグモイド関数の微分はどんな値を入れても0～0.25の値しかとらない。このため、勾配消失問題を起こす。

確認テスト



前述のとおり、正解は0.25の（3）である。

勾配消失の解決方法としては、①活性化関数を適切に選択する　②重みの初期値を工夫する　➂バッチの正規化　という3つの方法がある、以下それぞれについて詳細を説明する。

・活性化関数の選択

勾配消失問題を解決する活性化関数であるReLU関数を用いると、勾配消失問題を回避できる。ReLU関数は微分結果が0より小さい入力では0、0より大きい入力は1を出力する。入力が0より小さいか大きいかで、逆伝播を伝えるか伝えないかを切り替える。入力0以下は重みが使われないことになり、必要な重みのみが使われるようになる（スパース化）。

・重みの初期値を工夫する

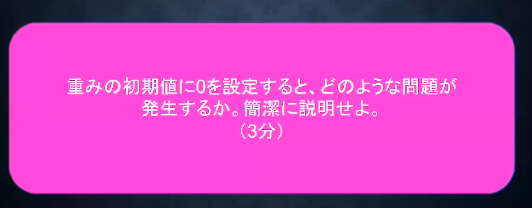
ニューラルネットワークを学習し始めるときに重みに与える初期値を工夫する。通常、入力された情報をいろんな見方をしながら学習をすすめたいので、初期値にはたいてい乱数を与える。しかし、重みはベクトルで表現されるが、そのベクトルに偏りがあると学習結果も偏ってしまう、あるいは学習がうまく進まないという問題が生じる。勾配消失問題を回避する乱数の設定方法にはいくつかあるが、有名なものにXavierの方法がある。通常ランダムな値には平均にたくさんの値があって、正規分布に従い分布する。正規分布（標準正規分布平均0分散1）にすると勾配消失問題を起こしやすい。そのため、正規分布の乱数に対して前の層のノード数のルートで割る。

サンプルコード演習

深栖学習day2補足資料.ipynbでXavier初期化を検証する。

標準正規分布で重みを初期化した場合、出力が0か1の値が頻出する分布となっている。この場合、シグモイド関数の微分をしたときに、0に収束する。逆伝播に向かない分布の分散を小さくすると、中間層の出力は0.5に集中する。この場合、どの層でも同じ値しか入っていない状態になり、学習がうまくいかない。Xavierの方法を使えば、0～1の値がまんべんのない頻度で出力され、極端な値の出力がなくなる。シグモイド関数に対してはうまくXavierの初期化が働く。ReLU関数に対してはHeの初期化が使われることが多く、√（2/前のノード数）で割る。標準正規分布で重みを初期化した場合、ReLU関数の微分の出力は0ばかりとなる。標準偏差を小さくしても表現力に変わりはない。Heの初期化により、0～1の値をまんべんなくとることが出来、ニューラルネットワークの表現力を維持ししながら、逆伝播の消失を回避し効率よく学習を進めることが出来る。

確認テスト

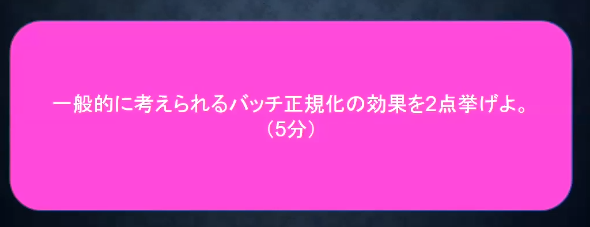


重みを0で初期化すると、すべての重みの値が均一に更新されるため、多数の重みをもる意味がなくなる。よって、正しい学習が行えなくなる。

・バッチの正規化

バッチの正規化は、ミニバッチ単位で、入力値のデータの偏りを抑制する方法である。ミニバッチは一回の学習計算量を意味する。GPUの場合ミニバッチは1～64、TPUの場合1～256である。コンピュータ上では1ミニバッチ分のデータしか見えていない。バッチ正規化を行うことで、中間層の重みの更新が安定化する。学習がうまくいくので高速化の効果がある。加えて、過学習を抑えることが出来る。方法としては、ミニバッチという単位で分布を正規化することで、学習データの極端なばらつきが抑えられる。ミニバッチ内の画像の平均を出し、その平均から分散を計算し、正規化する（統計処理と同じ要領である）。正規化された値を定数化したり、バイアスを加えたりしたりすることでニューラルネットワークが扱いやすい値とする。

確認テスト



・バッチ正規化を行うことで、中間層の重みの更新が安定化する。

・学習がうまくいくので高速化の効果がある。加えて、過学習を抑えることが出来る。

【勾配消失問題-実装演習】

サンプルコード演習

2\_2\_2\_vanishing\_gradient\_modified.ipynbのサンプルコードにおいて、

複数の中間層を持つニューラルネットワークを例とする。

vanishing sample

gauss分布に基づいてシグモイド関数で学習させる。勾配消失問題が発生してしまう例。

実際に学習結果をみると学習が全く行われていない。

また、ReLU関数に対して、gauss分布に基づいた重み初期化をした場合、ある所から急に学習が進む。

sigmoid関数に対してXavier初期化を行った場合、学習開始直後から精度が向上し、順調に学習が進んでいることがわかる。

ReLU関数に対してHe初期化を行った場合、も同様に学習開始直後から精度が向上し、順調に学習が進んでいることがわかる。

2\_3\_batch\_normalization.ipynbのサンプルコードにおいて、

BatchNormalizationのclassで、

実際のデータをミニバッチにして処理している。

mu=x.mean(axis=0)でミニバッチごとの平均をとり、

xc=x+muでｘをセンタリングし、

var=np,mean(xc\*\*2,axi=0)で分散を計算し、

std=np.sqrt(var+10e-7)でスケーリングを行い（標準偏差の計算）、

最後にxn=xc/stdとすることで、正規化処理が行われている。

これをニューラルネットワークのプログラムの中で実行されている。学習も順調に進んでいることがわかる。

コード確認テスト



（き）に当てはまる処理は、バッチサイズだけデータを取り出す処理なので、正解は、

data\_x[i:iend],data\_t[i:i\_end]

である。

【学習率の最適化手法】

勾配降下法では誤差の微分項に学習率をかけて引くという更新計算が行われている。このときの学習率の決め方には大きく4つ(モメンタム、AdaGrad、RMSProp、Adam)ある。

勾配降下法では、誤差関数の誤差の最小値を探索する。誤差率が大きすぎると、最小値にたどりつかず発散する。小さすぎると最小値に達するまでに時間がかかる。また、大局的極小解に収束しずらくなる（より小さい極小値にだどりつかない）。このため、学習率の最適化では、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。すなわちパラメータごとに学習率を可変にする。

以下では4つの学習率の最適化手法(モメンタム、AdaGrad、RMSProp、Adam)の詳細を説明する。

・モメンタム

モメンタムでは、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に、現在の重みに前回の重みを減残した値と慣性の積を加算する。前回の重みを使うので、最初は変化に乏しいが、学習が進むにつれて勾配に敏感に反応する。株価で言うところの移動平均のような変化をする（滑らかな動き）。メリットは局所的最適解にはならず、大局的最適解となる。また、谷間についてから最も小さい値（最適解）に到達するまでの時間が速い

・AdaGrad

AdaGradでは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。すなわち、これまでにどんな学習をしてきたか、を覚えながら更新量を決定する。メリットは誤差関数の勾配があまり深くなっておらず、緩やかな斜面に対して最適解に近づける。課題としては、学習率が徐々に小さくなるので。鞍点問題を引き起こす（大局的最適解に近づきづらい）。すなわち、傾きが平坦な部分では学習がうまく進みずらい（鞍点問題）。

・RMSProp

RMSPropでは、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。前回までの勾配情報どれくらい使うかを決めるパラメータを導入して、ＡｄａＧｒａｄを改良した方法となっている。メリットとしては、局所的最適解にはならず、大局的最適解となることが挙げられる。また、ハイパーパラメータの調整が必要な場合が少ない。

・Adam

Adamはモメンタムの、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の処理を両方含んだ最適化アルゴリズムである。メリットはモメンタム及びRMSPropのメリットをともに持ち合わせていることである。鞍点問題に陥りにくく、滑らかに誤差が小さくなっているので、学習を進めていってグラフを見ていると、学習がうまくいっているのかどうかの判断がしやすい。

【学習率最適化手法-実装演習】

サンプルコード演習

2\_4\_optimizer.ipynbのサンプルコードにおいて、

SGDの学習では、学習がうまくいかない、

モメンタムを使った学習でもうまくいかない。

RMSPropでは学習が順調に進む。

Adamも同様に学習が順調に進む。

このように、データが複雑な場合、学習がうまくいく最適化手法とうまくいかない最適化手法があり、最適化手法の選択は重要であることがわかる。

なお、今回のサンプルプログラムでは、正答率を評価しており、順調な学習の場合、学習が進むにつれて、正答率が上昇していく。

モメンタムのコード

V[key]=momentum\*v[key]-learning\_rate\*grad[key]および、network.params[key]+=v[key]という部分で更新量の計算を行っている。iが学習の回数でi>0すなわち二回目以降モメンタムによる更新処理が行われている。

Adagradのプログラム

h[key]=momentum\*h[key]-learning\_rate\*grad[key]

network.params[key]+=h[key]

の部分で、更新量の計算を行っている。

RMSPropのプログラム

新たにdecay\_rateというパラメータが設定されている。

h[key]\*=decay\_rate

h[key]+=(1-decay\_rate)\*np.square(grad[key])

network.params[key]-=learning\_rate\*grad[key]/(np.sqrt(h[key])+1e-7)

の部分で、更新量の計算を行っている。

【過学習】

過学習の状態では、検証時の誤差と訓練時の誤差に乖離がある。すなわち、検証用のデータに対してはあまり誤差が小さくならず、訓練データに対して過剰に誤差が小さい。学習が進んでいくにつれて、訓練用の誤差は小さくなっていくが、検証用データに対する誤差が小さくならなくなる。原因は入力値のデータが少ないのに、パラメータの数が多い、パラメータの値が適切ではない、ノードが多い等がある。つまりネットワークの自由度が高すぎることが原因である。自由度が上がるとより複雑な関数が表現できることが起因している。

【正則化手法】

過学習はニューラルネットワークの自由度が高すぎるのが原因で発生するので、正則化によって自由度を減らすことで、過学習を抑制する。Ｌ１正則化、Ｌ２正則化、ドロップアウトなどがある。

過学習の原因について詳細を説明する。

重みが極端に大きい値をとることで、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る。重みが大きすぎると1部の入力に対して、過大評価しており、極端な反応をしている。これがニューラルネットワークで発生している過学習の状況である。

続いて過学習の解決策について説明する。

誤差に対して、正則化項を加算することで、重みを抑制することが出来る。正規化項も加味して重みを調整する場合、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ばらつきを出す必要がある。すなわち、極端に大きい重みが発生しないようにする。

ここでpノルムは距離を意味し、誤差にノルムを加えるとき、以下の2つの方法がある。

・Ｌ2正則化：Ｐ２ノルム　ユークリット距離：二乗和√

・Ｌ１正則化：Ｐ１ノルム　マンハッタン距離：ただ足しただけで距離が求められる

Ｌ１正則化をラッソ回帰　Ｌ2正則化をリッジ回帰と呼ぶ。

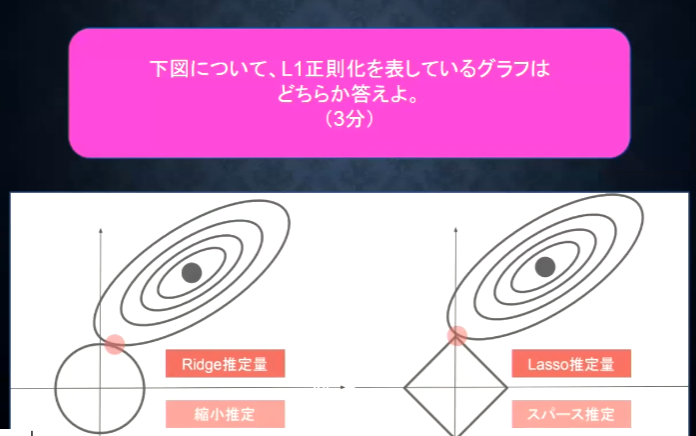
計算方法としては、ｐノルムは各々の重みをp上して和をとり、ｐ乗根を計算する。

誤差関数の誤差に、ハイパーパラメータとｐノルムと1/ｐをかけたものを足したのが正則化の計算となる。

正則化を実施するとき、正規化項のｐを変化させる。

誤差関数は通常、偏った位置に最小点がある。一方、正則化項は対称な位置に最小点がある。これらを合わせると最小点が少し、偏りない位置に修正される。Ｌ1ノルムだと、正則化項を適切に調整すれば、重みが0となる部分がある。この部分では重みが消去され、スパース化される効果がある。

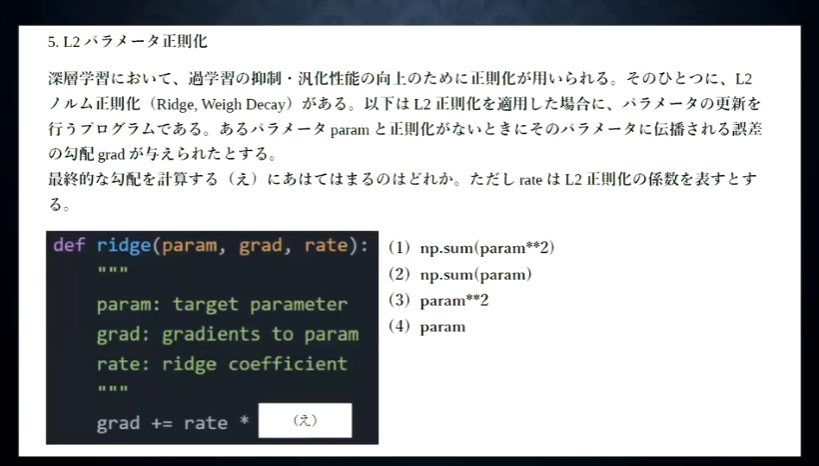
確認テスト



L1正則化は直線の等高線となるので、正解は右側のグラフである。1つの重みが0となる点を最適化で抽出されることからも右側のグラフが選べる。

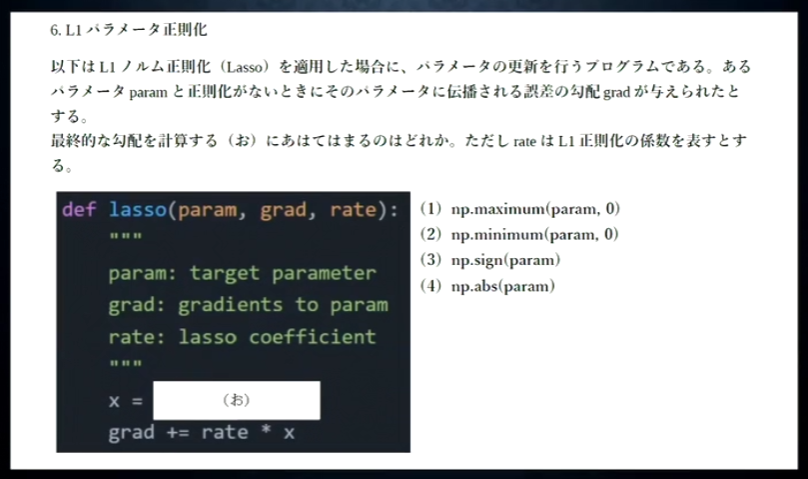
【過学習-例題】

確認テスト



L2正則化では、2乗の計算||param||^2がL2ノルムとなるので、その勾配が誤差の勾配に加えられる、つまり微分して誤差率の更新を行うときは、2\*paramとなるが、係数2は正則化の係数に吸収されても変わらない。よってparamが正解である。

確認テスト



L1正則化では、||param||がL1ノルムとなるので、その勾配が誤差の勾配に加えられる。つまり、np.sign(param)が正解となる。ここで、signは符号関数である。

・ドロップアウト

過学習の課題としてニューラルネットワークのノードの数が多いことが挙げられる。この課題を解決するために、ドロップアウトでは、ランダムにノードを削除して学習させる。

メリットは、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ことである。異なるネットワークで学習することになり、学習データにバリエーションが生まれる。

サンプルコード演習

2\_5\_overfiting.ipynbのサンプルコードにおいて、

overfitingという部分が過学習の評価プログラムである。

Weight-decay L2という部分でL2正則化が行われているプログラムが実装されている。

過学習のが起こる場合のプログラムでは、訓練データに対しては、精度100％となっているのに対し、テストデータに対しては、精度70％程度でとどまっており、過学習が発生していることがわかる。訓練データとテストデータで学習結果が乖離している。

L2正則化が行われているプログラムでは、テストデータに対しては、過学習が発生する例と比べて、精度が90％と少し下がる。これはL2正則化の効果によって、誤差関数が一番小さくなるポイントから少しずれたパラメータが採用されているためである。

L1正則化が行われているプログラムでは、1部の重みが消去されるので、特徴的な学習結果となる。L2正則化と同様テストデータに対しては、精度100％が取れていない状態となる。Dropoutが行われているプログラムでは順調に学習が進んでいる。ただし、先述のプログラムに比べ、学習の進み具合が遅くなっている。これは、ニューラルネットワークの中でデータ量が増えたような状態になったためである。

DropoutとL1正則化を両方適用したプログラムでは、両方の特徴が生かされ順調に学習が進む。

,

【ＣＮＮ】

畳み込みニューラルネットワーク（CNN）について説明する。CNNは画像の識別等でよく使われるが、それ以外にも、次元的なつながりがあるデータであれば、CNNで扱うことが出来る。次元的なつながりがあるデータとは、例えば画像で言うと緑の画素の近くは緑である可能性が高いということである。CNNは入力層（入力画像）、畳み込み層、プーリング層、全結合層、出力層から構成される。CNNの例としてＬｅＮｅｔがある。ＬｅＮｅｔは入力（32,32）→畳み込み演算、6個分の特徴量（28,28,6）→要約・サブサンプリング（14,14,6）→畳み込み演算、16個分の特徴量（10,10,16）→要約・サブサンプリング（5,5,16）→全結合層（120）→全結合層（84）→出力(10,)という構成である。

以下CNNで特徴的な層である、畳み込み層、パティングに関して詳細に説明する。

・畳み込み層

畳み込み層では入力値に対してフィルターをかけ出力値を得る。その後、出力層にバイアスを加え活性化関数を適用する。フィルターは全結合で言うところの重みに相当し。4×4の画像に対して、3×3のフィルターを適用すると2×2の出力が得られる。周りの情報のつながりを保ちながら、フィルター計算が行われる。これまでのニューラルネットワークでは全結合層で計算されていたところを。CNNでは畳み込みで計算が行われ、次元のつながりを保つ。畳み込み層が特徴量の抽出機の役割を果たしている。CNNの最後の全結合層では、人間が欲しい結果に変換する。畳み込み層では、画像の場合、縦、横、チャンネルの3次元のデータをそのまま学習し、次の層に伝えることが出来る。3次元の空間情報も学習できるような層が畳み込み層である。畳み込み演算では入力画像に対して、フィルターをスライドさせながら内積計算を実行する。畳み込み演算のあとはバイアス項を加えて最終的な出力にする。

・パティング

畳み込み演算をすると、元の画像データサイズが小さくなる。4×4の画像に対して、2×2のフィルターで読み込むと、出力は3×3となる。畳み込み演算を何回も繰り返すと、出力が小さくなりすぎてしまう。また、ＣＮＮ内でのデータサイズの整合が取れなくなる。この問題を解決しているのがパティングであり、上下左右に画像を広げる役割がある。パティングしたうえでフィルター処理すると、出力が元のデータサイズと同じになる。例として、0で埋めたり、近いデータと同じ値を入れる等の計算処理が行われる。。

・ストライド

畳み込み演算時のフィルターの移動はふつう1つずつだが、これを2つづつ移動する方法もある。前者はストライド1、後者はストライド2となる。

・チャンネル

フィルターの数を表す。

全結合で画像を学習した際のデメリットを説明する。画像の場合、縦、横、チャンネルの3次元で関連のあるデータだが、1次元データとして処理される。これら画像のデータ関連性が失われる。RGBの各チャンネル間の関連性が学習に反映されない。

【畳み込み層-実装演習】

サンプルコード演習

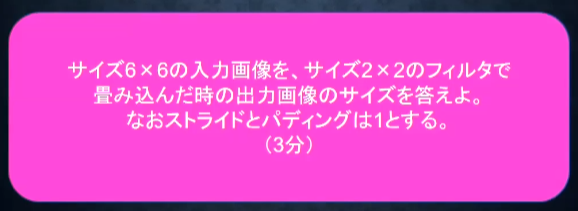
2\_6\_simple\_convolution\_network\_after.ipynbのサンプルコードにおいて、

プログラム上では畳み込み演算（フィルター処理）を高速に行えるように、Image2colという処理が行われている。Image2colではフィルターで切り取られる配列を1次元化して並べなおした1つの配列に変更される。Image2colによって1次元ベクトルの計算が可能となり計算コストが抑えられる。

【プーリング層】

プーリング層は畳み込み演算と組み合わせて使われることが多い。畳み込み演算と違って、プーリング層には重みがない。プーリング層にはMaxPoolingやAveragePoolingがある、それぞれ、対象領域の最大値、平均値を出力する。畳み込み演算と同様、計算対象区間をずらしながら、入力画像の配列に対し処理が行われる。

確認テスト



まず、入力画像をパテッィング数分を画像の両端へ増やす。そして、2×2フィルターサイズで、ストライド1（1つ飛ばし）で畳み込み演算を行うと、7×7の出力サイズとなる。

畳み込み演算の出力サイズに関して、以下の関係が成り立つので、この公式から求めてもよい。

[畳み込み演算の画像高さ]

＝（[入力画像の高さ]+2×[パティング]―[フィルターの高さ]）/[ストライド]+1

[畳み込み演算の画像幅]

＝（[入力画像の幅]+2×[パティング]―[フィルターの幅]）/[ストライド]+1

サンプルコード演習

2\_6\_simple\_convolution\_network\_after.ipynbのサンプルコードにおいて、

arg\_max=np.argmax(col,axis=1)

out=np.max(col,axis=1)

の部分でプーリング層をの処理が行われる。この例の場合、MaxPoolingで計算されている。また、Image2colで画像配列が1次元のベクトル化されているので、

out=out.reshape(N,out\_h,out\_w,C).transpose(0,3,1,2)

の部分で、元の画像サイズに戻している。

【AlexNet】

CNNの代表的なモデルである、AlexNetについて説明する。

AlexNet は5層の畳み込み層及びプーリング層など、それに続く3層の全結合層から構成される。詳細には以下の流れで演算が行われる。

1. 224×224の入力
2. 11×11のフィルタ96のチャンネル
3. 55×55の出力に対し、5×5のMAXPoolingをする256チャンネル
4. 27×27の出力に対し、3×3のMAXPoolingをする384チャンネル
5. 13×13の出力に対し、3×3のMAXPoolingをする384チャンネル（パティング処理も含まれる。）
6. 13×13の出力256のチャンネルに対し、全結合層3層で一列データに変換し出力層となる。最終的に1000個のデータ出力となる。

AlexNet ではFlatten処理をして、13×13×256のデータを横方向1列（43264個）のデータにする。ほかにもGlobal MaxPooling (13×13の中で一番大きい値を出力して256個の列データとする)やGlobal AvePooling(13×13の中で平均を出力して256個の列データとする)というプーリング処理が行われている。Global MaxPoolingとGlobal AvePoolingはFlattenにくらべデータ数が少なくなるが、Flattenに比べ効率的に特徴量を抽出でき、精度の良いモデルとなる。さらに、全結合層にドロップアウトを使用し、過学習を防いでいる。全結合層にドロップアウトが適用されることが多い