Abbreviation の約束
・FFN ; Feedforward Network

1　ネットワーク訓練

1.1　パラメタ最適化

ここからは、ネットワーク訓練に用いる誤差関数が与えられとき[*1]、その誤差関数を最小にする重みベクトル $\boldsymbol{w}$ を得るための手法を学んでいく。なお、議論を進める上で誤差関数が微分可能であることは断っておく。

まず、本サブセクションにおいては、我々が求めたい $\boldsymbol{w}$ を得るためにやらなければならないことを定義する。ここで理解して欲しいのは以下の 2 ポイントである。

・$\nabla E(\boldsymbol{w}) = 0$ なる停留点は 1 点に限定できないということ（一般的に大域的 -global- 極小値と局所的 -local- 極小値が存在[*2]）
・方程式 $\nabla E(\boldsymbol{w}) = 0$ を解析的に解くことは不可能であり数値的な反復手法が必要になること

2 つめのポイントを踏まえて、今回我々がとるアプローチは以下のようなアルゴリズムを導入することである。

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \Delta\boldsymbol{w}^{(\tau)} \tag{1}$$

ここで、τ はアルゴリズムのステップ数を表している。

我々がやることとしては「$\nabla E(\boldsymbol{w}) = 0$ という方程式を直接解くのではなく、$\boldsymbol{w}$ を動かして $\nabla E(\boldsymbol{w}) = 0$ となるような $\boldsymbol{w}^*$ を見つけてくる」というものである。したがって、我々のミッションはアルゴリズム（式 1）の各ステップにおいて、$\boldsymbol{w}$ が得られるたびに $\nabla E(\boldsymbol{w})$ の値が 0 に近いかどうかを評価するというものになる。

本サブセクション以降のストーリーは大まかに以下のようになる。

1. 5.2.2 では局所二次近似という手法を用いて、今回知りたい $E(\boldsymbol{w})$ の極小点を定義する[*3]。
2. 5.2.3 では誤差関数の極小点を探索するにあたり、勾配情報を用いるそもそものモチベーションを理解する[*4]。
3. 5.2.4 では式 1 における重み更新量を実用的な[*5]ネットワーク訓練方法にマッチした形で定義する。
4. 5.3 以降では重み更新量を効率よく評価する方法を導入し、どのような点で効率が良いのか理解する[*6]。

1.2　局所二次近似

ここでは上記ストーリーの 1. について、誤差関数 $E(\boldsymbol{w})$ に対してテーラー展開を施すところを起点に議論を行う。テーラー展開で 2 次の項までを取り出すという近似のもとで、極小点 $\boldsymbol{w}^*$ を定義するところまで辿り

[*1] ここでは誤差関数を一般的な表現で与えていることに注意してください。セクション 5.2 の導入においては、特定の誤差関数（二乗和誤差、交叉エントロピー誤差）を選んでくるという設定で説明がなされています。議論の前提が変わっていることに注意してください。
[*2] また、重み空間対称性によって等価な重みベクトルが多数存在することからも、停留値を限定できないという事情もあります。詳細は p.232 の 5.1.1 を参照。
[*3] 正直ここの位置付けが、これを書いてる 3/14 時点ではよくわかっていません。5.2.4 以降において、式 3 や式 7 を仮定して議論を進めているのかもしれませんが、これら 2 式を仮定することがどのように生きてくるのかがわからないのです。
[*4] ここは話がちょっと飛んでいると思います。個人的には 5.3 の導入（第一段落の次とか）に書くべきなんじゃないかなと思っていて、このレジュメではそのような建て付けで説明を行います。
[*5] 何をもって「実用的」なのかは該当サブセクションでちゃんと説明します。ご心配なく。
[*6] 本レジュメでは方法の導入までを議論します。効率の良さについては次回に譲るものとします。ちなみに、効率は計算負荷のオーダーという観点から議論されるものとなります。このことは 5.2.3 でも基本的なアイディアについては取り扱います。

着いてみせる。

テーラー展開（5.28）〜極小点周りでの局所二次近似の表式（5.32）の導出までは口頭で確認する。

ここからやりたいことは重み空間内における誤差関数の幾何学的イメージを理解することである。実は結論から言うと、局所二次近似を施した誤差関数の等高線は、極小点の近傍において楕円を描くことがわかる。

以下ではこの事実を導くために、その準備として重みベクトルについて線形和に展開していくことを考える。

まず（5.32）内のヘシアン[*7]について（5.33）のような固有方程式を考えることができる。ヘシアンが実対称行列であることから、固有ベクトルは正規直交基底となり[*8]、重みに関するベクトル $\boldsymbol{w} - \boldsymbol{w}^*$ を以下のような線形結合の形で表現できる。

$$\boldsymbol{w} - \boldsymbol{w}^* = \sum_i \alpha_i \boldsymbol{u}_i \tag{2}$$

式 2 を（5.32）へ代入すると

$$\begin{aligned}
E(\boldsymbol{w}) &\simeq E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \alpha_i \boldsymbol{u}_i^T \boldsymbol{H} \sum_j \alpha_j \boldsymbol{u}_j \\
&= E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j \boldsymbol{u}_i^T \boldsymbol{H} \boldsymbol{u}_j \\
&= E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j \boldsymbol{u}_i^T \lambda_j \boldsymbol{u}_j \ (\because (5.33)) \\
&= E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \sum_j \lambda_j \alpha_i \alpha_j \boldsymbol{u}_i^T \boldsymbol{u}_j \\
&= E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \sum_j \lambda_j \alpha_i \alpha_j \delta_{ij} \ (\because (5.34)) \\
&= E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2 \ (\because i = j \text{ のみ生存})
\end{aligned} \tag{3}$$

という誤算関数 $E(\boldsymbol{w})$ に対する近似表現[*9]を得ることができる。

それでは誤差関数の等高線が極小点の近傍において楕円を描くことを示していく（対応；演習 5.11）[*10]。方針としては楕円の方程式を露わな形で表現するということになる。まず、式 3 の最左辺と最右辺に注目する。

$$E(\boldsymbol{w}) = E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2 \tag{4}$$

[*7] ヘッセ行列とこれまで呼んでいましたが、ヘシアンの方が響きがクールなので、こっちを使っていきたいと思います。

[*8] 詳細は教科書 p.317-8 の説明を参照して欲しいです。

[*9] 本当にどうでもいい話ですが、私は一応近似のレベルによってニアリーイコール記号の種類を使い分けています。注目変数のオーダー（桁）のレベルで近似できるときは $\sim$、有限の次数までとってくるというときは $\simeq$ を使うようにするなど（例えばテーラー展開において 2 次までの項を取ってくることを考えよ）。ちなみに、$\fallingdotseq$ はあんまり使わない。

[*10] もちろん局所二次近似を施した上での話ですが。

ここで $E(\boldsymbol{w})$ が一定だから、これを C とおくと

$$
\begin{aligned}
C &= E(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2 \\
\frac{1}{2}\sum_i \lambda_i \alpha_i^2 &= C - E(\boldsymbol{w}^*) \\
\sum_i \lambda_i \alpha_i^2 &= 2(C - E(\boldsymbol{w}^*)) \\
\sum_i \lambda_i \alpha_i^2 &= \tilde{C} \ (\because \tilde{C} = 2(C - E(\boldsymbol{w}^*)))
\end{aligned} \tag{5}
$$

のように変形できる。なお、$\tilde{C}$ も定数である。これは変数 α_i に関する楕円の方程式になっており、軸の長さは式 5 において、特定の i 以外の α_j を全て 0 にすることで得られる。

$$
\begin{aligned}
\lambda_i \alpha_i^2 &= \tilde{C} \\
\alpha_i^2 &= \frac{\tilde{C}}{\lambda_i} \\
\alpha_i &= \sqrt{\frac{\tilde{C}}{\lambda_i}} \sim \sqrt{\frac{1}{\lambda_i}}
\end{aligned} \tag{6}
$$

軸の長さはヘシアンの固有値の平方根に反比例することがわかる（対応；図 5.6）。

重み空間が 1 次元の場合、停留点 w^* が以下のような条件を満たすときに、この停留点は極小点となる。

$$
\left.\frac{\partial^2 E}{\partial w^2}\right|_{w^*} > 0 \tag{7}
$$

このことは 1 次元の場合については自明である。[*11]したがって以降、我々はこのような定義を満たす極小点を見つけてくれば良いということになる。この後 2 つ先[*12]のサブセクション 5.2.4 では、$\boldsymbol{w}$ を動かす方法について、重み更新量をどのように定義するかというポイントで見ていくことになる。

1.3 勾配情報の利用

この内容は誤差逆伝播のセクションの冒頭で説明することにする。

1.4 勾配降下最適化

ここでは式 1 における重み更新量の設定方法として、オンライン勾配法と呼ばれる手法を学ぶ。これはネットワーク計算において一般的に採用されている手法となる。

まずオンライン手法と反対のアプローチとして位置付けられるバッチ手法についてアルゴリズムの式を確認する。アルゴリズムの式は以下の通りである。

$$
\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E(\boldsymbol{w}^{(\tau)}) \tag{8}
$$

[*11] 言い方が悪いですが、高校生でもわかります。

[*12] 5.2.3 との断絶があることは繰り返し申し上げておきます。

式 8 においては、右辺第二項の重み更新量を全データの誤差関数で計算している。実はこれは性能が悪い[*13]アルゴリズムとして知られていて、一般的には次のオンライン手法によるアルゴリズム式を使う。

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n(\boldsymbol{w}^{(\tau)}) \tag{9}$$

ここで $E_n(\boldsymbol{w}^{(\tau)})$ は

$$E(\boldsymbol{w}) = \sum_{n}^{N} E_n(\boldsymbol{w}) \tag{10}$$

を満たしている。この手法では重みベクトルの更新は新しいデータが 1 つ得られるたびに実行されていくことになる。この表式 9 からオンライン手法はバッチ手法に比べて以下の 2 点で優れていると言える。

・バッチサイズが膨大すぎてバッチでまとめて学習ができない場合でも逐次的に学習を行うことで計算負荷を小さくできる。

・1 つ 1 つのデータの影響を大きく受けるため、局所解にトラップされづらい[*14]。

したがって、我々は $\Delta E_n(\boldsymbol{w})$ を重み更新量として捉えたうえで、次のサブセクション 5.3 において議論を展開していくことにする。

2　誤差逆伝播

このセクションでは、誤差関数の重みに関する微分を評価する方法を導入し、その手法が計算論的に効率的であることを理解する。

ここからのストーリーは大まかに以下のようになる。

0. 誤差関数の極小点を探索するにあたり、勾配情報を用いるそもそものモチベーションを理解する[*15]。

1. 誤差関数が例えば二乗和誤差で与えられる場合について、$\nabla E_n(\boldsymbol{w})$ が入力や出力を用いてどのように表現できたかを振り返る。

2. 上記例を一般の FFN において議論し、$\nabla E_n(\boldsymbol{w})$ が隠れユニット変数を用いた一般的な表式で書けることを学ぶ。

3. 出力ユニットや隠れユニットにおける活性化関数を具体的に定め、誤差関数の重みに関する微分が一般的な表式に従うことを確認する[*16]。

それでは上記ストーリー 0. について、計算負荷のオーダーに注目して議論を進める。わけわからん。$\boldsymbol{b}$ と $\boldsymbol{W}$ から極小点を求めるために $O(W^2)$ だけの計算負荷になることだけは理解できた。極小点それぞれの評価において $O(W)$ だけのステップが必要になる、という考え方が全く理解できていない状態。そして $\nabla E_n(\boldsymbol{w})$ を評価するごとに W 個の情報が得られるという意味もわからない。

[*13] どういう意味で性能が悪いと言っているのかよくわかりませんが、計算の手間がかかるという意味であれば、いざ訓練計算を始めたときに 1 エポックぶん終わらせるのに、全データを使わなければならないということがあります。または、悪い結果が得られてしまうという意味であれば、局所解に陥ってしまうとそこから動けなくなってしまうということがあります。一方で、この後紹介するオンライン手法はこの 2 点を解決した手法となっています。

[*14] 逆を言えば外れ値が入ってきたときに、その値に大きく引っ張られて学習が進むということにもなります。オンライン手法も当然万能ではありません。折衷案としてバッチ手法とオンライン手法のバランスを取ったミニバッチ手法なるものがありますが、教科書の説明では出てこないため、このレジュメでもパスさせていただきます。

[*15] 教科書では 5.2.3 の内容に対応しています。このレジュメでは説明の都合のため、ここで議論することにします。

[*16] ここは次回に譲ります。

2.1 誤差関数微分の評価

ここでは、上記ストーリー 1. と 2. について、ネットワーク内における情報の流れる向きに注目して考えていく。特に、ネットワークの順向きと逆向きに情報を流すという「逆伝播」という操作を定義し、これが微分の評価に用いられることを理解する。

以降の議論では、セクション 5.2.4 でも定めたように、対象とする誤差関数は 1 つのデータ点についての誤差 $E_n(\boldsymbol{w})$ であることを前提とする。

一般的な FFN について考える前に、まず誤差関数が例えば（5.46）のように二乗和誤差で与えられるとき、勾配が（誤差）×（入力変数）という形式で得られることを確認する。ここで、y_{nk} は出力であり (5.45) を満たす。また、t_{nk} は目標変数である。（5.46）を重み w_{ji} で微分すると

$$
\begin{aligned}
\frac{\partial E_n}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \\
&= \frac{\partial}{\partial y_{nj}} \frac{\partial y_{nj}}{\partial w_{ji}} \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \\
&= \frac{\partial}{\partial y_{nj}} \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} x_i \ (\because (5.45)) \\
&= \frac{1}{2} \cdot 2 \sum_k (y_{nk} - t_{nk}) \frac{\partial y_{nk}}{\partial y_{nj}} \sum_i \delta_{ji} x_i \\
&= \sum_k (y_{nk} - t_{nk}) \delta_{kj} x_i \ (\because j = i \text{ のみ生存}) \\
&= (y_{nj} - t_{nj}) x_i \ (\because k = j \text{ のみ生存}) \qquad (11)
\end{aligned}
$$

確かに誤差関数の微分が（誤差）×（入力変数）という形式で表現されている。この結果が、誤差関数の形を特定しない一般の FFN において成立することを確かめていく。この成立を示すためのフレームワークは次の通り。

・議論の準備として[*17]順伝播の考え方について理解する。
・今回我々が欲しい E_n の勾配を求めるために、重み成分 w_{ji} での偏微分を行う。
・連鎖公式を巧み[*18]に用いることで、勾配が隠れユニットの誤差を使った形（5.56）で表せることを導く。
・実は隠れユニットにおける誤差が出力ユニットにおける誤差を用いて表せることを確かめる（逆伝播というアイディアの導入）。
上記フレームワークに従い、（5.56）を導いてみせる。

まず、準備として順伝播の定義を確認する。順伝播はこれまで取り扱ってきたような入力→出力を（5.48）と（5.49）を用いてネットワーク計算していくものである。一方で、このサブセクションのオチとして、出力→入力という順番で計算するということが起こる（逆伝播）。後者の結果を導くこともこのサブセクションの 1 つのアウトプットである。

ネットワーク計算の考え方について理解してもらったところで、我々の目的としている誤差関数の勾配を考

[*17] この後に出てくる逆伝播との違いを理解するための準備として、という意味です。
[*18] 後知恵感は否めません。

えていく。シンプルに誤差関数 E_n を重み成分 w_{ji} で微分する。

$$\begin{aligned}\frac{\partial E_n}{\partial w_{ji}} &= \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \\ &= \delta_j \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} z_i \ (\because (5.51), (5.48)) \\ &= \delta_j \sum_i \delta_{ji} z_i \\ &= \delta_j z_i \ (\because j = i \text{ のみ生存}) \end{aligned} \tag{12}$$

δ_j が何なのかまだよくわかっていない[*19]が、ひとまず式 12 の形を見ると（誤差）×（入力変数）という形になっていそうである。

次にこの δ_j がどのように表されるかを考えていく。δ_j は（5.51）のように定義されており、ここから変形を試みる。

$$\begin{aligned}\delta_j &= \frac{\partial E_n}{\partial a_j} \\ &= \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \ (\because \text{Chain rule}) \end{aligned} \tag{13}$$

一応これがちゃんと δ_j に戻ることを確かめる。

$$\begin{aligned}\sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} &= \sum_k \frac{\partial E_n}{\partial a_k}\delta_{kj} \\ &= \frac{\partial E_n}{\partial a_j} \ (k = j \text{ のみ生存}) \\ &= \delta_j \end{aligned} \tag{14}$$

確かめ終わり。

再び式 13 から議論を始める。

$$\begin{aligned}\delta_j &= \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\ &= \sum_k \delta_k \frac{\partial a_k}{\partial a_j} \ (\because (5.51) \text{ において } j \to k \text{ とした}) \\ &= \sum_k \delta_k \frac{\partial}{\partial a_j} \sum_i w_{ki} z_i \ (\because (5.48) \text{ において } j \to k \text{ とした}) \\ &= \sum_k \delta_k \frac{\partial}{\partial a_j} w_{kj} z_j \ (\because i = j \text{ のみ生存}) \\ &= \sum_k \delta_k w_{kj} \frac{\partial}{\partial a_j} h(a_j) \ (\because (5.49)) \\ &= h'(a_j) \sum_k w_{kj} \delta_k \end{aligned} \tag{15}$$

式 15 が言っていることは、出力側（k）の δ の総和によって入力側（j）[*20]の δ が表せるということである。そして、これがネットワークにおいて逆伝播が起こっているということである。なお、ユニットの値を計算す

[*19] 残念なことに、これが誤差だとわかるのは次のサブセクション 5.3.2 にて具体例を取り扱うときになります。

[*20] 正確には出力より手前側の隠れユニットです。

る際に順方向と逆方向で総和の取り方が異なっていて、前者は添字の 2 文字目[*21]について、後者は添字の 1 文字目[*22]についてそれぞれ和を取るという違いがあることに注意せよ。

以上より、誤差関数の微分に対する評価方法のフレームワークは p.246 上部のようにまとめられる。出力ユニットから隠れユニットの誤差が得られ、そしてこれが誤差関数の微分の中身に現れるため、∇E を評価したことになる。この評価が可能になることは今回我々がゴールとして設定していたことである。

[*21] （5.48）を見よ。
[*22] （5.56）を見よ。