6.1 例: XOR の学習

フィードフォワードネットワークの概念をより具体的にするために, 非常に単純なタスク XOR 関数の学習を行う完全に機能するフィードフォワードネットワークの例から始める.

XOR 関数 (「排他的論理和」) は, 2 つのバイナリ値 x_1 と x_2 に関する演算である. これらのバイナリ値の 1 つが 1 に等しい場合, XOR 関数は 1 を返す. それ以外の場合は 0 を返す. XOR 関数は, 学習したいターゲット関数 $y = f^*(\boldsymbol{x})$ を提供する. 我々のモデルは関数 $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ を提供し, 我々のアルゴリズムはパラメータ $\boldsymbol{\theta}$ を変化させて f を可能な限り f^* に近づける.

この単純な例では, 統計的一般化には関心がない. ネットワークが 4 つの点 $\mathbb{X} = \{[0,0]^\top, [0,1]^\top, [1,0]^\top, [1,1]^\top\}$ で正しく動作することを望む. これら 4 つの点全てでネットワークを訓練する. 唯一の課題は訓練セットを適合させることである.

この問題を回帰問題として扱えば, 平均二乗誤差損失関数関数を使用できる. この例の計算を可能な限り単純化するために, この損失関数を選択する. バイナリデータをモデル化するためのより適切な他のアプローチがあることはのちに説明する.

トレーニング全体で評価すると, MSE 損失関数は次のようになる.

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2. \tag{6.1}$$

次に, モデルの形式 $f(\boldsymbol{x}; \boldsymbol{\theta})$ を選択する必要がある.$\boldsymbol{\theta}$ が $\boldsymbol{w}$ と b で構成される線形モデルを選択するとする. モデルは次のように定義される.

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b. \tag{6.2}$$

正規方程式を使用することで, $\boldsymbol{w}$ および b に関して閉じた形式で $J(\boldsymbol{\theta})$ を最小化できる.

正規方程式を解くと, $\boldsymbol{w} = \boldsymbol{0}$ および $b = \frac{1}{2}$ を得る. 線形モデルは単純にどこでも 0.5 を出力する. なぜこのようなことが起こるのか? 図 6.1 は線形モデルが XOR 関数を表現できないことを示している. この問題を解決する 1 つの方法は, 線形モデルが解を表現できる別の特徴空間を学習するモデルを使用することである.

図 6.1: 表現を学習することによる XOR 問題の解決.
プロット上に表示された太字の数字は, 学習された関数が各点で想定される出力値を示す. (左) 元の入力に直接適用された線形モデルでは, XOR 関数を実装できない. $x_1 = 0$ の場合, x_2 が増加するにつれてモデルの出力も増加する. $x_1 = 1$ の場合, x_2 が増加するにつれてモデルの出力が減少する. 線形モデルでは, 固定係数 w_2 を x_2 に適用する必要がある. したがって, 線形モデルは x_1 の値を使用して x_2 の係数を変更することはできず, この問題を解決できない. (右) ニューラルネットワークによって抽出された特徴によって表現される返還された空間では, 線形モデルが問題を解決できるようになる. この解決例では, 出力 1 を持つ必要のある 2 つの点が, 特徴空間の 1 つの点に折りたたまれている. つまり, 非線形特徴は, $\boldsymbol{x} = [1,0]^\top$ と $\boldsymbol{x} = [0,1]^\top$ の両方を特徴空間内の単一点 $\boldsymbol{h} = [1,0]^\top$ にマッピングする. 線形モデルは, h_1 で増加し, h_2 で減少する関数を記述することができる. この例では, 特徴空間を学習する誘因は, 訓練セットに適合できるようにモデルの容量を大きくすることだけである. より現実的なアプリケーションでは, 学習された表現はモデルの一般化にも役立つ.

具体的には, 2 つの隠れユニットを含む 1 つの隠れ層を持つ非常に単純なフィードフォワードネットワークを導入する. このモデルの図については, 図 6.2 を参照. このフィードフォワードネットワークには, 関数 $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ によって計算される隠れユニット $\boldsymbol{h}$ というベクトルがある. これらの隠れユニットの値は, 2 番目の層の入力として使用される. 2 番目の層はネットワークの出力層である. 出力層は依然として単なる線形回帰モデルだが, $\boldsymbol{x}$ ではなく $\boldsymbol{h}$ に適用されるようになった. ネットワークは 2 つの関数 $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ と $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$ が連結し

たものを含んでおり, 完全なモデルは $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)})(\boldsymbol{x})$ である.

図 6.2: 2 つの異なるスタイルで描かれたフィードフォワードネットワークの例.
具体的には, これは XOR の例を解くために使用するフィードフォワードネットワークである. 2 つのユニットを含む 1 つの隠れ層がある. (左) このスタイルでは, 全てのユニットをグラフ内のノードとして描画する. このスタイルは非常に明示的で明確だが, この例よりも大規模なネットワークの場合に大量の空間をとる可能性がある. (右) このスタイルでは, 層の活性化を表すベクトル全体のノードをグラフに描画する. このスタイルはより簡潔である. 場合によっては, このグラフのエッジに 2 つの層の間の関係を説明するパラメータの名前を注釈としてつけることがある. ここで, 行列 $\boldsymbol{W}$ が $\boldsymbol{x}$ から $\boldsymbol{h}$ へのマッピングを記述し, ベクトル w が $\boldsymbol{h}$ から y へのマッピングを記述することを示す. 通常, この種の図面にラベルをつける場合は, 各層に関連づけられた切片パラメータを省略する.

$f^{(1)}$ はどの関数を計算する必要があるのか? これまでのところ, 線形モデルはうまく機能しており, $f^{(1)}$ も線形にしたいという誘惑に駆られるかもしれない. 残念ながら, $f^{(1)}$ が線形の場合, フィードフォワードネットワーク全体は入力の線形関数のままになる. 当面は切片項を無視して, $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^\top \boldsymbol{x}$ および $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^\top \boldsymbol{w}$ と仮定する. すると, $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{W}^\top \boldsymbol{x}$ となる. この関数は, $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w'} (\boldsymbol{w'} = \boldsymbol{W}\boldsymbol{w})$ として表すことができる.

特徴を記述するには, 明らかに非線形関数を使用する必要がある. ほとんどのニューラルネットワークは, 学習されたパラメータによって制御されるアフィン変換と, それに続く活性化関数と呼ばれる固定の非線形関数を使用して, これを実行する. ここでは, $\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}$ を定義することでその戦略を用いる. ここで, $\boldsymbol{W}$ は線形変換の重みを提供し, $\boldsymbol{c}$ はバイアスを提供する. 以前は線形モデルを記述するために, 重みのベクトルとスカラーのバイアスパラメータを使用して, 入力ベクトルから出力スカラーへのアフィン変換を記述した. ここで, ベクトル $\boldsymbol{x}$ からベクトル $\boldsymbol{h}$ へのアフィン変換を説明するため, バイアスパラメータのベクトル全体が必要になる. 活性化関数 g は通常, 要素ごとに適用される関数として選択される ($h_i = g(\boldsymbol{x}^\top \boldsymbol{W}_{:,i} + c_i)$). 最新のニューラルネットワークでは, デフォルトの推奨は, 活性化関数 $g(z) = \max\{0, z\}$ で定義される ReLU(Rectified Linear Unit) を使用することである. ReLU は, Jarrett ら (2009), Nair と Hinton(2010), Glorot ら (2011a) によって定義されており, 図 6.3 に示される.

図 6.3: 修正された線形活性化関数
この活性化関数は, ほとんどのフィードフォワードニューラルネットワークでの使用が推奨されるデフォルトの活性化関数である. この関数を線形変換の出力に適用すると, 非線形変換が得られる. ただし, この関数は, 2 つの線形部分を持つ区分線形関数であるという意味で, 線形に非常に近いままである. 修正された線形ユニットはほぼ線形であるため, 勾配ベースの方法で線形モデルを簡単に最適化できる多くの特性が保持される. また, 線形モデルを適切に一般化するための多くの属性も保存される. コンピュータサイエンス全体に共通する原則は, 最小限の構成要素から複雑なシステムを構築できるということである. チューリングマシンのメモリは 0 または 1 の状態を保存できれば良いのと同じように, 修正された線形関数から汎用関数近似器を構築できる.

これで完全なネットワークを次のように指定できるようになった.

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b. \tag{6.3}$$

これで, XOR 問題の解決策を指定できるようになった. 仮に,

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \tag{6.4}$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \tag{6.5}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \tag{6.6}$$

$b = 0$ としよう.

ここで, モデルが入力のバッチを処理する方法を見てみる. $\boldsymbol{X}$ をバイナリ入力空間内の 4 つの点全てを含む計画行列とし, 行ごとに 1 つの例を示す:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{6.7}$$

ニューラルネットワークの最初のステップは, 入力行列に第 1 層の重み行列を乗算することである:

$$\boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}. \tag{6.8}$$

次にバイアスベクトル $\boldsymbol{c}$ を追加して以下を得る

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \tag{6.9}$$

この空間では, 全ての例が傾き 1 の線に沿って配置されている. この線に沿って移動すると, 出力は 0 から始まり, 1 に上昇し, その後 0 に戻る必要がある. 線形モデルではそのような関数を実装できない. 各列の $\boldsymbol{h}$ の値の計算を完了するには, 修正された線形変換を適用する:

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \tag{6.10}$$

この変換により, サンプル間の関係が変化した. それらは単一の直線上にはない. 図 6.1 に示すように, それらは線形モデルが問題を解決できる空間に位置している.

最後に重みベクトル w を乗算する:

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \tag{6.11}$$

ニューラルネットワークは, バッチ内の全ての例に対して正しい答えを取得した.

この例では, 単純に解を指定し, 誤差が 0 であることを示した. 実際の状況では, 数十億のモデルパラメータと数十億の訓練例が存在する可能性があるため, ここで行ったように単純に解決策を推測することはできない.

代わりに, 勾配ベースの最適化アルゴリズムを使用すると, 誤差がほとんど発生しないパラメータを見つけることができる. XOR 問題に対して説明した解決策は損失関数の大域最小値あるため, 勾配降下法はこの点に収束する可能性がある. XOR 問題には, 勾配降下法でも見つけられる同等の解決策が他にもある. 勾配降下法の収束点はパラメータの初期値に依存する. 実際には, 勾配降下法では通常, ここで示したような綺麗で理解しやすい整数値の解は見つからない.

+———————————————-+
要約
XOR の学習は非常に単純なタスクである. XOR 関数は, 2 つの入力が異なる場合に 1 を返し, 等しい場合に 0 を返す. ここで, 単純な線形モデルでは XOR を学習することができない. それは, 線形モデルでは 2 つの入力の組み合わせに対して単一の直線しか表現できないためである. XOR を学習するためには, 非線形なモデルが必要になる. 一般的には, ニューラルネットワークが使用される. ニューラルネットワークは複数の層に渡って非線形変換を適用することで, 複雑な関数をモデル化できる. XOR の場合, 少なくとも 1 つの隠れ層を持ったニューラルネットワークを使用すると XOR を学習することができる.
+———————————————-+