4.5 重量崩壊

オーバーフィットの問題を特徴づけたので、モデルを正規化するための標準的な手法をいくつか紹介します。外出してより多くのトレーニングデータを収集することで、常にオーバーフィッティングを軽減することができることを思い出してください。費用がかかる、時間がかかる、または完全に私たちの手に負えなくなることがあります。

短期的には不可能になる今のところは、リソースが許す限りの高品質なデータをすでに持っていると仮定して、正則化技術に焦点を当てることができます。

多項式曲線フィットの例(セクション 4.4)では、フィットした多項式の次数を微調整するだけでモデルの容量を制限することができます。確かに、特徴の数を制限することは、オーバーフィッティングを避けるための一般的なテクニックです。しかし、単に機能を脇に捨てるだけでは、仕事のためにあまりにも鈍いハンマーになってしまうことがあります。多項式の曲線フィットの例に固執して、高次元の入力で何が起こるかを考えてみましょう。多変量データへの多項式の自然な拡張は単項式と呼ばれ、単に変数の累乗の積である。単項式の次数は、その累乗の和である。例えば、x21 x2 、x3 x25 は共に次数 3 の単項式である。

次数 d の項の数は、d が大きくなるにつれて急激に増えていくことに注意してください。k 個の変数が与えられると、次数 d の単項式の数は H(k, d) となります。程度の小さな変化、例えば 2 から 3 への変化でさえ、モデルの複雑さを劇的に増大させます。このように、機能の複雑さを調整するためには、より細かなツールが必要になることが多いのです。

4.5.1 二乗ノルム正則化

重み減衰（一般的に L2 正則化と呼ばれる）は、パラメトリック機械学習モデルを正則化するために最も広く使われている手法かもしれません。この手法は、すべての関数 f の中で、関数 f = 0（すべての入力に値 0 を代入する）がある意味で最も単純であり、ゼロからの距離によって関数の複雑さを測定できるという基本的な直感に基づいています。

しかし、関数とゼロの間の距離をどのように正確に測ればよいのでしょうか？正解は一つではありません。実際、関数解析やバナッハ空間の理論を含む数学の全分野が、この問題に答えるために費やされています。

単純な解釈としては、線形関数 f (x) = w⊤ x の複雑さを重みベクトルのノルム、例えば ||w||2 ののような、で測定することが考えられます。小さな重みベクトルを確保するための最も一般的な方法は、損失を最小化する問題に、そのノルムをペナルティ項として加えることです。このように、学習ラベルの予測損失を最小化するという当初の目的を、予測損失とペナルティ項の和を最小化するという新たな目的に置き換えます。さて、重みベクトルが大きくなりすぎると、学習アルゴリズムは学習誤差の最小化よりも重みノルムの最小化に焦点を当てることになるかもしれません。それこそが、私たちが求めているものです。コードで説明するために、以前の例をセクション 3.1 の線形回帰のために復活させてみましょう。

そこで、私たちの損失は次のように与えられていました

(4.5.1)

x(i) はオブザベーション，y (i) はラベル，(w, b) はそれぞれ重みとバイアス・パラメータであることを思い出してください．重みベクトルのサイズにペナルティを与えるためには、何らかの方法で損失関数に $|||w||2$ を加えなければなりませんが、モデルはこの新しい加算ペナルティに対して標準損失をどのようにトレードオフすべきでしょうか？実際には、このトレードオフを正則化定数 $\lambda > 0$（検証データを用いて適合させる非負のハイパーパラメータ）を介して特徴づけます。

(4.5.2)

$\lambda=0$ の場合は、元の損失関数を回復します。$\lambda > 0$ の場合は、$||w||$ の大きさを制限します。鋭い読者は、なぜ標準ノルム（ユークリッド距離）ではなく二乗ノルムを使うのか疑問に思うかもしれません。これは計算上の便宜を図るためです。L2 ノルムを 2 乗することで、重みベクトルの各成分の 2 乗の和を残して平方根を除去します。これにより、ペナルティの微分が簡単に計算できるようになります（微分の和は和の微分に等しくなります)。

さらに、そもそもなぜ L1 ノルムではなく L2 ノルムを使うのかと疑問に思うかもしれません。
実際には、他の選択肢は統計学全体で有効であり、人気があります。L2 正則化線形モデルが古典的なリッジ回帰アルゴリズムを構成していますが、L1 正則化線形回帰は統計学において同様に基本的なモデルです（一般的にはラッソ回帰として知られています)。
一般的には、$\ell 2$ は p ノルムと呼ばれる無限のノルムの一つであり、将来的には多くのノルムに遭遇することになるだろう。一般に、ある数 p に対して、ℓ p ノルムは次のように定義される。

(4.5.3)

L2 ノルムを使用する理由の 1 つは、重みベクトルの大きな成分に大きなペナルティを課していることです。これにより、学習アルゴリズムは、より多くの特徴に均等に重みを配分するモデルに偏ります。実際には、これは単一の変数での測定誤差に対してよりロバストなものになるかもしれません。対照的に、L1 のペナルティは、他の理由で望ましいかもしれない小さな特徴のセットに重みを集中させるモデルをもたらします。
L2 正則化回帰のための確率的勾配降下更新は以下の通りである。

(4.5.4)

前述のように、我々の推定値と実測値の差に基づいて w を更新する。
しかし、w の大きさも 0 に向かって縮小していきます。そのため、この方法は「重み減衰」と呼ばれることがあります。ペナルティ項だけを与えられた場合、最適化アルゴリズムはトレーニングの各ステップで重みを減衰させます。特徴選択とは対照的に，重み減衰は，f.の複雑さを調整するための連続的なメカニズムを提供してくれます． λ の値が小さいと制約のない w に対応し、λ の値が大きいとかなり w が制約されます。対応するバイアスペナルティ b 2 を含めるかどうかは、実装によって異なり、ニューラルネットワークの層によって異なる場合があります。しばしば、ネットワークの出力層のバイアス項を正則化しないことがあります。

4.5.2 高次元線形回帰
簡単な合成例を用いて、特徴選択よりも重みの減衰の利点を説明することができます。まず、以前と同じようにデータを生成します。

(4.5.5)

ラベルが入力の線形関数になるように選択され、平均値が 0、分散が 0.01 のガウスノイズによって破損します。オーバーフィッティングの影響を顕著にするために、問題のディメンシノリティを d = 200 に増加させ、わずか 20 の例を含む小さなトレーニングセットで作業することができます。
4.5.3 スクラッチからの実装
次に、元の目標関数に単純に 2 乗 ℓ 2 のペナルティを加えるだけで、ゼロから重み減衰を実装していきます。
モデルパラメータの初期化
まず、モデルパラメータをランダムに初期化する関数を定義し、それぞれに対して attach_grad を実行して、計算する勾配のメモリを確保します。

ℓ 2 ノルムペナルティの定義
おそらく、このペナルティを実行するための最も便利な方法は、すべての項をその場で二乗して合計することです。私たちは慣習的に 2 で割っています（2 次関数の微分を取るとき、2 と 1/2 は相殺され、更新のための式がきれいでシンプルに見えることを保証します)。

トレーニング機能とテスト機能の定義

以下のコードは、学習集合にモデルを適合させ、テスト集合で評価します。線形ネットワークと二乗損失は前章から変わっていないので、d2l.linreg と d2l.squared_loss を使ってインポートするだけです。ここでの唯一の変更点は、我々の損失がペナルティ項を含むようになったことです。

定型化されていないトレーニング
今、lambd=0 でこのコードを実行し、重量減衰を無効にします。オーバーフィットが悪いので、学習誤差は減少しますが、テスト誤差は減少しないことに注意してください。

重み減衰の使用
以下、しっかりと重み減衰を実行する。トレーニングエラーは増加しますが、テストエラーは減少することに注意してください。これはまさに正則化に期待する効果です。練習問題として、重り w の $\ell 2$ ノルムが実際に減少していることを確認してみましょう。

4.5.4 簡潔な実装
ニューラルネットワークの最適化では、重みの減衰はどこにでも存在しているため、Gluon は特に便利で、最適化アルゴリズム自体に重みの減衰を統合し、任意の損失関数と組み合わせて簡単に使用できます。さらに、この統合は計算上の利点を提供しており、追加の計算オーバーヘッドなしにアルゴリズムに重み減衰を追加する実装トリックを可能にします。更新の重み減衰部分は各パラメータの現在値にのみ依存するため、オプティマイザはいずれにせよ各パラメータに一度は触れる必要があります。
以下のコードでは、Trainer のインスタンスを作成する際に wd を介して重み減衰ハイパーパラメータを直接指定しています。デフォルトでは、Gluon は重みとバイアスを同時に減衰させます。モデルパラメータを更新する際には，ハイパーパラメータ wd に wd_mult が乗算されることに注意してください．したがって、wd_mult を 0 にすると、バイアスパラメータ b は減衰しません。

このプロットは、我々がゼロから重量減衰を実装したときのものと同じに見えます。しかし、彼らはかなり速く実行され、実装が容易になり、大きな問題のためにより顕著になる利点があります。

これまで、単純な一次関数を構成するものについて、一つの概念にしか触れていませんでした。さらに、単純な非線形関数を構成するものは、さらに複雑な問題になります。例えば、カーネルヒルベルト空間の再現（RKHS）は、線形関数のために導入されたツールを非線形コンテキストで適用することを可能にします。残念なことに、RKHS ベースのアルゴリズムは、純粋に大規模で高次元のデータだけにスケールする傾向があります。この本では、ディ

ープネットワークの全層に重み減衰を適用するという単純なヒューリスティックをデフォルトにします。

要約 - 正規化は、オーバーフィッティングに対処するための一般的な方法です。これは、学習モデルの複雑さを軽減するために、学習集合の損失関数にペナルティ項を追加します。
- モデルをシンプルにするためには、$\ell 2$ のペナルティを用いて重みを減衰させることが重要である。これは、学習アルゴリズムの更新ステップでの重みの減衰につながる。
- Gluon はハイパーパラメータ wd を設定することで、オプティマイザの自動重量減衰機能を提供しています。
- 同じトレーニングループ内で、異なるパラメータセットに対して異なるオプティマイザを持つことができます。

Exercises 1. Experiment with the value of λ in the estimation problem in this page. Plot training and test accuracy as a function of λ. What do you observe?
2. Use a validation set to find the optimal value of λ. Is it really the optimal value? Does this matter?
Σ
3. What would the update equations look like if instead of $\|w\|2$ we used i $|wi|$ as our penalty of choice (this is called $\ell 1$ regularization).
4. We know that $\|w\|2 = w^\top w$. Can you find a similar equation for matrices (mathematicians call this the Frobenius norm74)?
5. Review the relationship between training error and generalization error. In addition to weight decay, increased training, and the use of a model of suitable complexity, what other ways can you think of to deal with overfitting?
6. In Bayesian statistics we use the product of prior and likelihood to arrive at a posterior via $P(w \mid x) \propto P(x \mid w)P(w)$. How can you identify $P(w)$ with regularization?