CS231n Convolutional Neural Networks for Visual Recognition

線形分類

前のセクションでは，画像の分類問題を紹介しました．さらに，学習セットの(注釈付きの)画像と比較することで画像にラベルを付けるk-近傍法(kNN)分類器について説明しました．これまで見てきたように，kNNには数々の欠点があります．

> ・分類器は学習データをすべて記憶し，将来のテストデータとの比較のために保存しなければなりません．データセットのサイズが容易にギガバイト級になるため，容量効率が悪くなる．
> ・テスト画像の分類は，すべてのトレーニング画像との比較を必要とするため，コストがかかる．

この章の概要を説明に入ります．私たちは今，画像分類に対するより強力なアプローチを開発しようとしていますが，これは最終的にはニューラルネットワークや畳み込みニューラルネットワーク(CNN)全体にまで拡張していくことになります．このアプローチには2つの主要なコンポーネントがあります：生(raw)データをクラスのスコアにマッピングするスコア関数と，予測されたスコアと正解ラベル(教師データ)の間の一致を定量化する損失関数です．次に，これを最適化問題として，スコア関数のパラメータに関して損失関数を最小化します．

画像からラベルスコアへのマッピングをパラメタライズする

このアプローチの最初の構成要素は，画像のピクセル値を各クラスの信頼度スコアにマッピングするスコア関数を定義することです．具体的な例を挙げてアプローチを展開していきます．

前述のように，画像 $x_i \in R^D$ の学習データセットを仮定して，それぞれがラベル y^i に関連付けられているとします．(但し，$i = 1...N, y_i \in 1...K$とする)
つまり，N個のサンプル(それぞれ次元数D)とK個の異なるカテゴリがあります．例えば，CIFAR-10では，10の異なるクラス(犬，猫，車…など)があるため，それぞれ次元数$D = 32 \times 32 \times 3 = 3072$ピクセル，カテゴリ数$K = 10$，訓練画像数 $N = 50{,}000$のセットとなります．
ここでスコア関数を定義します．

$$f : R^D \mapsto R^K$$

この関数は生の画像のピクセルをクラスのスコアにマップするものです．

線形分類器． このモジュールでは，可能な限り最も単純な関数である線形写像から始めます．

$$f(x_i, W, b) = Wx_i + b$$

上記の式において，画像 x_i は，その全画素が形状[$D \times 1$]の1列ベクトルに再配列されていると仮定している．行列 W(= [$K \times D$]サイズ) とベクトル b (=[$K \times 1$]サイズ) は，この関数における変数です．

Wのパラメータはしばしば「重み」と呼ばれます．bはバイアスベクトルと呼ばれます．出力スコア(出力段)に影響を与えるものの，データx_i自体には影響を与えないためです．俗に「パラメータ」と言った場合は「重み」のことを指します．

注意点:

・単一行列の乗算 Wx_i は，各分類器が行列Wの行方向の10個それぞれの分類器(各クラスに1つ)を並列に評価していることに注意してください．

・また，入力データ(x_i, y_i)は固定のものと考えていますが，パラメータW, bの設定は制御できます．目標は，計算されたスコアが学習セット全体の正解ラベル(教師信号)と一致するように，これらのパラメータを設定することです．これがどのように行われるかについてはもっと詳しく説明しますが，感覚的には，正しいクラスのスコアは不正確なクラスのスコアよりも高くなる事を期待できます．

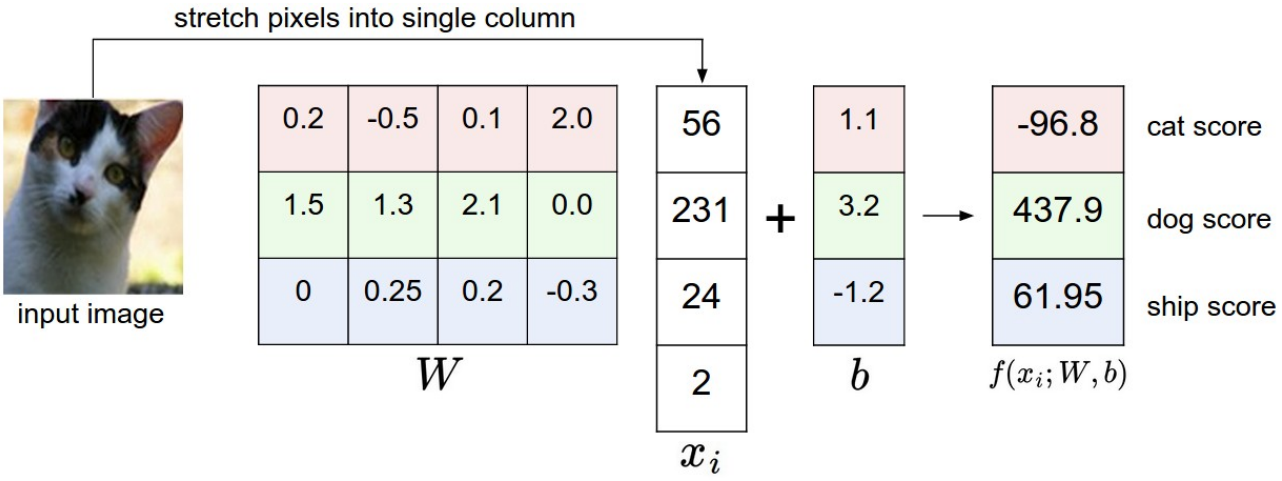
・テスト画像の分類には，1つの行列の乗算と加算が含まれており，テスト画像をすべてのトレーニング画像と比較するよりもかなり高速です

先取り解説:

畳み込みニューラルネットワークは，上記のように画像ピクセルとスコアを正確にマッピングしますが，マッピング関数(f)はより複雑で，より多くのパラメータを含みます．

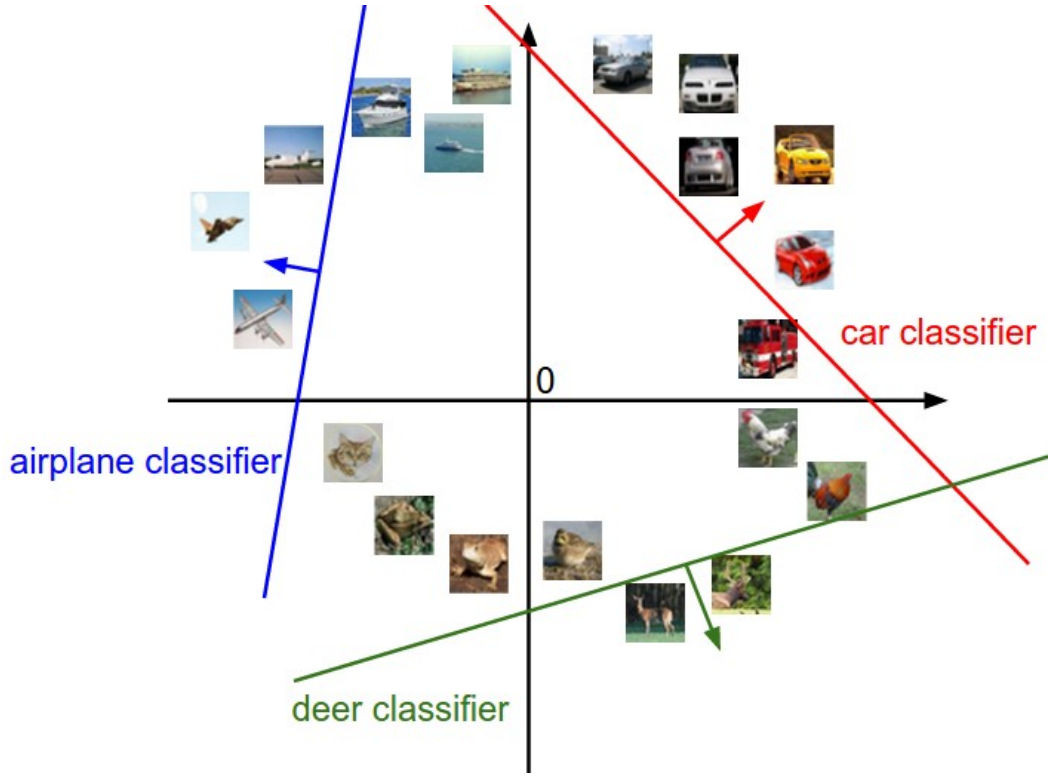
線形分類器の解析

線形分類器は，あるクラスのスコアを，RGBカラーチャンネルのすべてのピクセル値の重み付けされた合計として計算します．これらの重みにどのような値を設定するかによって，この関数は画像の特定の位置にある特定の色を(それぞれの重みの符号に応じて)好き嫌いする能力を持っています．例えば，画像の側面に青が多い場合(これは水に対応している可能性が高い)，「船」クラスの可能性が高いと想像することができます．この場合，「船」分類器は青チャンネルの重みには正の重みが多く(青の存在は船のスコアを増加させます)，赤/緑チャンネルの重みには負の重みが多く(赤/緑の存在は船のスコアを減少させます)，と予想するかもしれません．



上は画像と分類スコアの対応付けの例である．画像x_iのピクセル数は モノクロ4 ピクセル (この例では簡潔にするためにカラーチャンネルは考慮していません) としている．3 つの分類 (赤 (=猫)，緑 (=犬)，青 (=船) のクラス) があると仮定します．(単に3つの分類を色分けして示しているだけであり，RGBチャンネルとは関係ありません．念のため)．画像のピクセルを1列に伸ばし，各クラスのスコアを得るために行列の乗算を行う．しかしここで取り上げている重みWのセットは全く良好な判定をしていない．重みWは，猫の画像であるにもかかわらず猫の分類へ低い判定を下しており，画像は犬であると判定している．

高次元の点としての画像の類推．画像は高次元の列ベクトルに引き伸ばされているので，各画像はこの空間における1つの点として解釈できます (例えば，CIFAR-10の各画像は，32x32x3ピクセル=3072次元空間における点とみなせる)．同様に，データセット全体はラベル付きの点の集合である．
各クラスのスコアをすべての画像ピクセルの重み付き和として定義したので，各クラスのスコアはこの空間上の線形関数となります．3072次元の空間を可視化することはできませんが，これらの次元をすべて2次元上に押しつぶすことを想像すると，分類器が何をしているのかを可視化することができます．



上の図は各画像を1点とし，3つ(航空機・車・鹿)の分類器を可視化した画像空間の視覚的表現．車の分類器を例にすると(赤線)，赤線は，車のクラスのスコアが0になる空間内のすべての点を示している．赤い矢印は増加方向を示しているので，赤線の右側にある点はすべて正の(直線的に増加する)スコアを持ち，赤線の左側にある点はすべて負の(直線的に減少する)スコアになる．

上記から分かる通り，Wの各行はクラスの1つの分類器です．これらを幾何学的に考えると，Wの行の1つを変えることによってピクセル空間の対応する線が異なる方向に回転するということです．一方，バイアスbを変更すると，分類器は線を平行移動させます．特に，バイアス項がない場

合，$x_i = 0$を入力すると，重みに関係なく常に0となり，すべての分類器の線が原点を遠らざるを得ないことに注意してください．

線形分類器のテンプレートマッチングとしての解釈．重みWのもう一つの解釈は，Wの各行がクラスの1つのテンプレート(またはプロトタイプと呼ばれることもある)に対応するということです．画像に対する各クラスのスコアは，内積(またはドット積)を用いて各テンプレートを画像と比較し，最も「フィットする」ものを見つけることによって得られます．この用語を用いて表現するのであれば，線形分類器はテンプレートを学習する「テンプレートマッチング」を行っていることになります．別の考え方としては，効果的に最近傍探索を行っていますが，(kNNのように)何千もの学習画像を持つ代わりに，クラスごとに1つの画像を使用しているだけです．また，L1やL2の距離の代わりに，(負の)内積を距離として使用します．

ちょっと先取り：上記はCIFAR-10の学習終了時に学習した重みの例．例えば，船のテンプレートには，予想通り青いピクセルが多く含まれています．そのため，このテンプレートは，海の上の船の画像との内部積を照合すると，高いスコアが得られます．

上図の馬のテンプレートには双頭の馬が含まれているように見えますが，これはデータセットの左向きと右向きの馬の両方が含まれているためです．線形分類器は，データ中の馬のこれら2つのモードを1つのテンプレートに統合します．同様に，車の分類器は，いくつかの車種や向きといったモードを結合して1つのテンプレートにしたようです．また，CIFAR-10データセットには，他のどの色の車よりも赤い車が多いことも示唆しています．線形分類器は色の異なる車を適切に識別には弱すぎますが，後に見るように，ニューラルネットワークはこのタスクを実行できるようにしてくれます．少し先の話ですが，ニューラルネットワークは，隠れた層で特定の車のタイプ(例えば，左向きの緑の車，前向きの青の車など)を検出する中間層ニューロンを開発することができ，次の層のニューロンは，個々の車検出器の重み付けされた合計を通して，これらをより正確な車のスコアに結合することができるようになるでしょう．

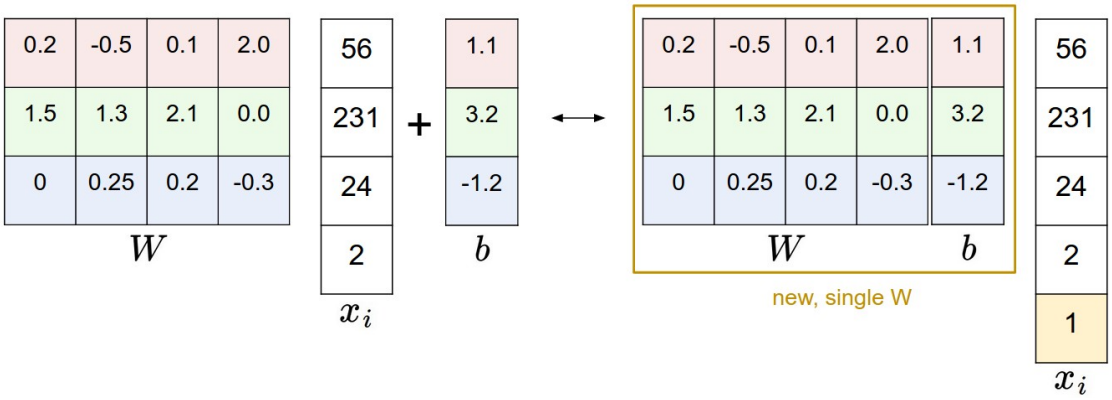
バイアスのトリック．先に進む前に，2つのパラメータW, bを1つのものとして表現するための一般的な単純化のトリックについて触れておきたいと思います．スコア関数を次のように定義したことを思い出してください．

$$f(x_i, W, b) = Wx_i + b$$

2 つのパラメータセット(バイアス b と重み W)を別々に追跡するのは少し面倒です．一般的に使用されるトリックは，ベクトルx_iを，常に定数 1 –デフォルトのバイアス次元を保持する1 つの次元–を追加して，2 つのパラメータセットを 1 つの行列に結合することです．次元が追加されると，新しいスコア関数は単一の行列の乗算に単純化されます．

$$f(x_i, W) = Wx_i$$

CIFAR-10の例では，x_iは$[3072 \times 1]$の代わりに$[3073 \times 1]$になり(追加した次元が定数1を保持しています)，W は$[10 \times 3072]$のではなく$[10 \times 3073]$となりました．W が現在のバイアスbに対応する追加された列は，下図のように図示することで明確になるかもしれません．



バイアストリックの図解. 行列の乗算を行ってからバイアスベクトルを加算すること(左)は，すべての入力ベクトルx_iに定数1のバイアス次元を追加し，重み行列Wにバイアス列bを1列拡張することと等価です(右). このように，すべてのベクトルに1を加えることでデータを前処理する場合，重みとバイアスを保持する2つの行列の代わりに，重みの1つの行列を学習するだけでよい.

画像データの前処理. 上記例では，生のピクセル値([0...255]からの範囲)を使用しました．機械学習では，入力特徴(画像の場合，すべてのピクセルが特徴として考えられます)の正規化を常に実行することが一般的です．特に，すべての特徴から平均値を引くことでデータの中心を決めることが重要です．画像の場合，これはトレーニング画像全体の平均画像を計算し，すべての画像からそれを減算して，ピクセルが約[-127 ... 127]の範囲にある画像を得ることに相当します．さらに一般的な前処理として，各入力特徴量の値が[-1, 1]からの範囲になるようにスケーリングすることがあります．データ全体がゼロを中心に分布させる正規化も重要です．詳細は勾配降下法を取り上げるときに扱います.

損失関数

前節では，ピクセル値から分類スコアへの関数を定義しました．(これは前節では固定値でした) しかし，この「重み」を制御することし，予測されたクラスのスコアが学習データの正解ラベル(教師データ)と一致するように設定したいと考えています.

例えば，猫の画像とクラス "cat", "dog", "ship "のスコアの例に戻ると，この例ので使用した重みのセットはあまり良くないことがわかります．私たちは猫を描いたピクセルに餌を与えましたが，猫のスコアは他のクラス (犬のスコア 437.9 と船のスコア 61.95) に比べて非常に低い (-96.8) でした．このような結果に対する私たちの不幸の度合を，損失関数(コスト関数や目的関数とも呼ばれることがある)で測定しようとしています．感覚的には，学習データの分類がうまくいっていなければ損失は大きくなり，うまくいっていれば低くなります.

マルチクラスサポートベクターマシンにおける損失関数.
損失関数の詳細を定義する方法はいくつかあります．最初の例として，まずマルチクラスサポートベクターマシン(SVM)損失と呼ばれる一般的に使用される損失を開発します．SVM 損失は，SVM が各画像の正しいクラスが不正確なクラスよりも一定のマージンΔだけ高いスコアを持つことを

「望む」ように設定されています．上で行ったように，損失関数を擬人化することが役に立つことがあります．SVMは，その結果がより低い損失(これは良いことです)をもたらすという意味で，ある結果を「望んでいる」のです．
ここで，より正確に理解してみましょう．例ではi番目の画像x_iのピクセルと，正解クラスのインデックスを指定するラベルy_iが与えられています．スコア関数は，ピクセルを受け取り，クラスのスコアのベクトル$f(x_i, W)$を計算します．これをsとします．例えば，j番目の分類のスコアはj番目の要素となり，このように表せます： $s_j = f(x_i, W)_j$. 次に，i番目の例のマルチクラスSVM損失は，以下の式で表されます．

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

例を挙げます．どのように動作するかを見てみましょう．スコア$s = [13, -7, 11]$を受け取る3つのクラスがあり，最初のクラスが真のクラス (すなわち$y_i = 0$)であるとします．また，Δ(=ハイパーパラメータ:近々詳しく説明します)が10であると仮定します．上の式は，すべての不正なクラス(=真ではないクラス:$j \neq y_i$)を合計するので，以下の様に2つの項が得られる．

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

$[-7 - 13 + 10]$が負数になるので，最初のmax(0,−)関数はゼロになることがわかります．正解の分類のスコア(13)が不正解のクラスのスコア(-7)よりも少なくともマージンは10($= \Delta$)より大きいので，このペアの損失はゼロになります．実際の差は20であり，マージンの10よりもはるかに大きいですが，SVMは差が少なくとも10あることだけに留意します．次に第2項は[11 - 13 + 10]=8です．つまり，正しい分類($S_{y_j=0} = 13$)は不正確な分類($S_{j=2} = 11$)よりも高いスコアを持っていたにもかかわらず (13 > 11)，それは期待されているマージン，10より大きくありません．差が僅か2なので，損失は8です．(つまり，マージンを満たすためには差がより大きくなければならない)．
要約すると，SVM損失関数は，正しい分類y_iのスコアS_{y_i}が不正確な分類のスコアよりも少なくともΔだけ大きくなることが望まれます．そうでない場合は，損失を蓄積することになります．
このモジュールでは，線形スコア関数 $f(x_i; W) = Wx_i$ を扱っているので，この等価な形式で損失関数を書き換えることもできます．

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

ここでw_jは，Wのj番目の行を列として転置したものです．しかし，より複雑なスコア関数fの形を考え始めると，必ずしもこうなるとは限りません．

この節を終える前に最後に言及する用語は，max(0,-)関数のゼロ時のしきい値です．しばしばヒンジ損失(ヒンジ関数)と呼ばれます．SVMヒンジ損失(L1-SVM)を使う代わりに，$\max(0,-)^2$で表される，SVM二乗ヒンジ損失(L2-SVM)という形式を使用しているという話を聞くことがあります．こ

れは，違反したマージンをより強く(線形ではなく二次的に)ペナルティを与えるものです．二乗しないヒンジ損失はより標準的ですが，いくつかのデータセットでは，二乗ヒンジ損失を用いた方がよりよく機能することがあります．これは，クロス・バリデーション中に決定できるものです．

☞ 損失関数は，訓練セットの予測値に対する我々の"不満足度"を定量化します．

マルチクラスSVMは，正しい分類のスコアが他のすべてのスコアよりも少なくともΔマージンで高くなることを「望んで」います．もし，あるクラスのスコアが赤い領域の内側にある(またはそれ以上の)場合，累積損失が発生します．そうでなければ，損失はゼロになります．我々の目的は，学習データ中のすべての例について，同時にこの制約を満たし，可能な限り，低いトータル損失になる重みWを見つけることです．

正則化．上で紹介した損失関数には1つのバグがあります．データセットと，すべての例を正しく分類するパラメータWの集合があるとしましょう(すなわち，すべてのスコアがすべてのマージンを満たすように，すべてのiについて$L_i = 0$であるとします)問題は，このWの集合が必ずしも一意であるとは限らないということです．例を正しく分類する類似のWがたくさんあるかもしれません．これを見る簡単な方法の1つは，あるパラメータがすべての例を正しく分類している場合(したがって，損失は各例についてゼロ)，これらのパラメータに任意数を掛けたλWも損失はゼロになります．なぜなら，この変換はすべてのスコアの大きさを均一に拡大しただけなので，その絶対的な差も伸びるからです．例えば，正しいクラスに最も近い不正解のクラスの間の得点の差が15であった場合，Wのすべての要素に2をかけると，新しい差は30になります．
言い換えれば，この曖昧さを取り除くために，ある重みWのセットに対する変化を符号化したいと考えています．これは，損失関数を正則化ペナルティ$R(W)$で拡張することで実現できます．
最も一般的な正則化ペナルティはL2ノルムで，すべてのパラメータに対して要素ごとに2次ペナルティを与えることで，大きな重みを抑制します．

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

上の式では，Wのすべての2乗要素を合計しています．正則化関数はデータの関数ではなく，重みだけに基づいていることに注意してください．正則化ペナルティを含めることで，データ損失(全ての例の平均損失L_i)と正則化損失の2つの成分からなる完全なマルチクラスSVMの損失関数が完成します．つまり，完全なマルチクラスSVM損失となるのです．

$$\underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

展開すると以下の形になります

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

ここで，Nは学習例の数です．ご覧のようにハイパーパラメータλで重み付けされた正則化ペナルティを損失関数に追加します．このハイパーパラメータを設定する簡単な方法はなく，通常はクロスバリデーションによって決定されます．

上記で述べた理由の他にも正則化ペナルティを含めることが望ましい特性がたくさんあり，その多くは後のセクションで説明します．例えば，L2ペナルティを含めることで，SVMにおける良好な最大マージン特性が得られることがわかります(詳細はCS229の講義ノートを参照してください．訳註：リンク切れ?)

この特性の最も魅力的な面は，大きな重みにペナルティを与えると一般化が改善される傾向があるということです．どの入力次元もそれだけではスコアに非常に大きな影響を与えることができなくなるからです．例えば，入力ベクトル$x = [1,1,1,1]$と2つの重みベクトル$w_1 = [1,0,0,0]$, $w_2 = [0.25,0.25,0.25,0.25]$があるとします．すると，$w_1^T x = w_2^T x = 1$なので，重みベクトル両者は同じドット積になるが，$w_1$のL2ペナルティは1.0であるのに対し，$w_2$のL2ペナルティは0.25にしかなりません．したがって，L2 のペナルティによれば，重みベクトルw_2の方が正則化の損失が少なくて済むので好ましいと思われます．感覚的に言えば，w_2の重みがより小さく，より拡散しているといえるからです．L2ペナルティは，より小さく拡散性の高い重みベクトルが好ましいので，最終的な分類器は，入力次元の要素のうちの一部を非常に大きくするのではなく，入力次元の全ての要素を考慮し，それぞれを小さくするすることを推奨します．この授業の後半で説明するように，この効果は，テスト画像に対する分類器の一般化性能を向上させ，オーバーフィッティングの減少につながります．

バイアスは，重みとは異なり，入力次元の影響力の強さを制御しないので，同じ効果を持たないことに注意しましょう．したがって，一般的に正規化は重みWのみに行い，バイアスbには行いません．この影響はほとんどの場合，無視できるほどしかありません．最後に，正則化のペナルティにおいて，次の点に留意してください．すべての例で正確に0.0の損失を達成することはできません．それができるのは重み$W = 0$の時の異常状態の時だけです．

(次ページに続きます)

コードです．ここでは，Pythonで実装された損失関数(正則化なし)を，非ベクトル化と半ベクトル化の両方の形で示しています．

```
def L_i(x, y, W):
  """
  非ベクトル化バージョン．単一の例(x,y)についてのマルチクラスSVM損失を計算します．
  - x :画像を表す列ベクトル(CIFAR-10 では 3073 x 1)3073 番目の位置にバイアス次元が追
  加されています．
  - y :正しい分類のインデックスで，整数です (例: CIFAR-10 では 0 から 9 の間)．
  - W :重み行列(例えばCIFAR-10では10×3073)
  """
  delta = 1.0 # see notes about delta later in this section
  scores = W.dot(x) # scores becomes of size 10 x 1, the scores for each
class
  correct_class_score = scores[y]
  D = W.shape[0] # number of classes, e.g. 10
  loss_i = 0.0
  for j in range(D): # iterate over all wrong classes
    if j == y:
      # skip for the true class to only loop over incorrect classes
      continue
    # accumulate loss for the i-th example
    loss_i += max(0, scores[j] - correct_class_score + delta)
  return loss_i
def L_i_vectorized(x, y, W):
  """
  この関数はより高速動作するよう，ハーフベクトル化された実装です．
  ハーフベクトル化とは，1つの例に対して実装がforループを含まないことを意味します．
  しかし，(この関数の外に)まだ一つのループが残っています．
  """
  delta = 1.0
  scores = W.dot(x)
  # compute the margins for all classes in one vector operation
  margins = np.maximum(0, scores - scores[y] + delta)
  # on y-th position scores[y] - scores[y] canceled and gave delta. We want
  # to ignore the y-th position and only consider margin on max wrong class
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i

def L(X, y, W):
  """
    完全ベクトル化された実装．
  - X :すべての学習例を列として保持(例: CIFAR-10では3073×50,000)
  - y :正しいクラスを指定する整数の配列 (例: 50,000次元の配列)
  - W :重み(例: 10 x 3073)．
  """
  # evaluate loss over all examples in X without using any for loops
  # left as exercise to reader in the assignment
```

このセクションでは，SVMの損失は，学習データ上の予測値が正解ラベル(教師データ) とどの程度一致しているかを測定するために，ある特定のアプローチを取るということを示しています．さらに，訓練セットで良い予測を行うことは，損失を最小化することと同等です．

☞ 私たちが今しなければならないことは，損失を最小限に抑えるウェイトを見つける方法を思いつくことです

実践と考察

デルタの設定．ハイパーパラメータΔとその設定をする事について改めて触れましょう．どのような値に設定すべきか，また，クロスバリデーションを行う必要があるでしょうか？このハイパーパラメータは，すべてのケースにおいて$\Delta = 1.0$に設定しても安全であることがわかりました．ハイパーパラメータΔとλは2つの異なるハイパーパラメータのように見えますが，実際にはどちらも同じトレードオフを制御しています．データ損失関数と正則化損失関数のトレードオフです．

これを理解する上で重要なことは，重みWの大きさがスコアに直接影響を与えるということです(したがって，その差も)．Wの全ての要素の値を小さくすればスコアの差は小さくなり，Wを大きくするとスコアの差はすべて大きくなります．したがって，スコア間のマージンの厳密な値(例えば，Δ=1やΔ=100)は，この場合はあまり意味がありません．なぜなら，重みWは差を任意に縮小させたり増大させたりすることができるからです．したがって，唯一の本当のトレードオフは，(正則化の強さλを通して)重みをどれだけ大きくするかということです．
バイナリSVMとの関係．バイナリSVMはここまで学んだことを踏まえて以下の様に記述できるかもしれません．

$$L_i = C \max(0, 1 - y_i w^T x_i + R(W))$$

ただし，Cはハイパーパラメータであり，$y_i \in \{-1, 1\}$です．ここで提示した式は、分類が2つしかない場合の特殊なケースとしてバイナリSVMを含んでいることに納得できるでしょう。つまり、もし分類が2つしかない場合、損失は上で示したバイナリSVMに減少します。また、式中におけるCと今まで扱っていたλは同じトレードオフを制御しており、それぞれのパラメータの関係は $C \propto \frac{1}{\lambda}$ を満たします。

閑話休題。最適化の基本。もしあなたがSVMの知識を持ってこのクラスに来ているならば、カーネル、二重項、SMOアルゴリズムなどについても聞いたことがあるかもしれません。このクラスでは（一般的なニューラルネットワークの場合と同様に）常に制約のない原始形での最適化目的を扱います。これらの目的の多くは、技術的には微分可能ではありませんが（例えば、max(x,y)関数は、x=yのときに不連続点があるため、微分可能ではありません）、実際にはこれは問題ではなく、「劣微分」を使用するのが一般的です。

閑話休題2。他のマルチクラスSVM定式化。 本節で紹介するマルチクラスSVMは、複数の分類にわたってSVMを定式化する数少ない方法の一つであることは注目に値します。もう一つの一般的に使用されている形式は、One-VS-All (OVA) SVMで、各分類-対-他の全ての分類に対して独立したバイナリSVMを学習するものです。関連していますが、実際にはあまり一般的ではありませんが、All-vs-All (AVA) 戦略もあります。我々の定式化は、Westone and Watkins 1999 (pdf) のバージョ

ンに従っていますが、これはOVAよりも強力なバージョンです(このバージョンではデータ損失がゼロになるようなマルチクラスのデータセットを構築できますが、OVAではできません。興味のある方は、論文の詳細をご覧ください)最後のSVM定式化は、構造化SVMで、正しいクラスのスコアと最もスコアの高い不正解の次点クラスのスコアとの間のマージンを最大化するものです。これらの定式化の違いを理解することは、この授業の範囲外です。これらのノートで提示されたバージョンは、実際に使用すると安全に使用できる事が分かりますが、最も単純なOVA戦略も、同様にうまくいく可能性が高いです(Rikinら 2004年. “In Defense of One-VS-All Classification” (pdf))。

ソフトマックス分類器

SVMはよく見かける2つの分類器のうちの1つであることがわかりました。もう1つは、損失関数が異なるソフトマックス分類器が一般的です。以前に2値ロジスティック回帰分類器知っている方へ説明すると、ソフトマックス分類器はその複数のクラスへの一般化です。
出力$f(x_i, W)$を各クラスのスコアとして扱う SVM とは異なり（校正されておらず、解釈が難しいかもしれません）ソフトマックス分類器は、より直感的な出力（正規化されたクラス確率）を提供します。ソフトマックス分類器では、スコアマッピング関数$f(x_i; W) = Wx_i$は変化しませんが、これらのスコアを各クラスの正規化されていない対数確率として解釈するようにし、損失関数としてヒンジ損失ではなく、以下の式で表される交差エントロピー損失を使用します。

$$L_i = -\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \text{ or equvalentry } \quad L_i = \log \sum_j e^{f_j}$$