CS231n Convolutional Neural Networks for Visual Recognition

3.最適化:確率勾配降下法

序論

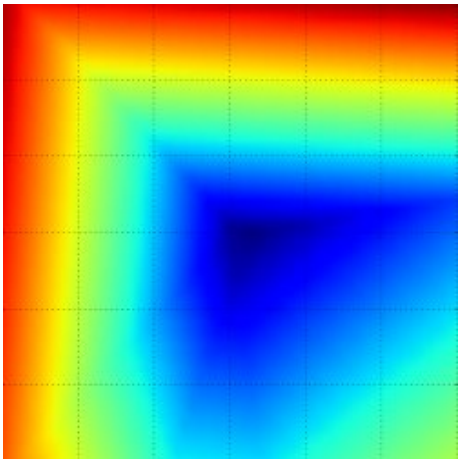
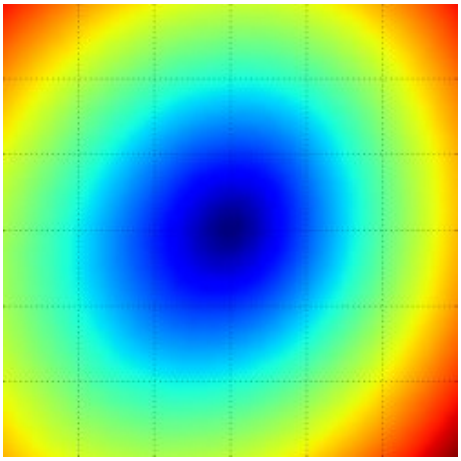
前のセクションでは，画像分類タスクの文脈で2つの重要な要素を紹介しました.
・生の画像のピクセルをクラスのスコアにマッピングする（パラメータ化された）スコア関数（例：線形関数).
・誘導されたスコアが訓練データの正解ラベル(教師データ) とどれだけ一致しているかに基づいて，特定のパラメータセットの品質を測定する損失関数.
これには多くの方法やバージョンがあることがわかりました（例：Softmax/SVM）．具体的には，線形関数が$f(x_i, W) = Wx_i$として定式化され，開発したSVMは

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)y_i) + 1)] + \alpha R(W)$$

例x_iの予測値が基底真理ラベルy_iと一致するようなパラメータWの設定では，損失Lが非常に小さくなることがわかりました．ここで，3番目で最後の重要な要素を紹介します．**最適化**です．最適化とは，損失関数を最小化するパラメータWのセットを見つけるプロセスです.
ちょっと先取り：これら3つのコアコンポーネントがどのように相互作用するかを理解したら最初の構成要素（パラメータ化された関数の写像）を再検討し，線形写像よりもはるかに複雑な関数に拡張していきます．最初にニューラルネットワーク全体，次に畳み込みニューラルネットワークにです．損失関数と最適化プロセスは相対的にみて変更されません.

損失関数の可視化

この授業で見ていく損失関数は，通常，非常に高次元の空間（例えば，CIFAR-10では，線形分類器の重み行列のサイズは[10 x 3073]で，合計30,730個のパラメータ）で定義されているため，可視化するのが難しいです．しかし，線（1次元）や平面（2次元）に沿って高次元空間をスライスすることで，1つの空間についていくつかの感覚的理解を得ることができます． 例えば，ランダムな重み行列W（空間内の1点に対応）を生成し，線に沿って進み，途中で損失関数の値を記録することができます．すなわち，ランダムな方向W_1を生成し，異なるaの値について$L(W + aW_1)$を評価することにより，この方向に沿った損失を計算することができ,この処理により，aの値をx軸，損失関数の値をy軸とする単純なプロットが生成されます．また，a, bを変化させたときの損失$L(W + aW_1 + bW_2)$を評価することで，2次元でも同じ手順を行うことができます．プロットでは，a,bはx軸とy軸に対応し，損失関数の値を色の変化で表現しています.

CIFAR-10におけるMulticlass SVM(正則化なし)の1つの例(左,中)と100個の例(右)の損失関数の表面図．左: a を変化させただけの一次元の損失． 中，右: 二次元の損失スライス，青 = 低損失，赤 = 高損失．損失関数の断片的な線形構造に注目してください．複数の例の損失は平均値と結合しているので，右のボウル形状は多くの区分線形関数のボウル（中央のようなもの）の平均値です．

損失関数の断片的な線形構造は，数学的に調査することで説明できます．一つの例を挙げると

$$L_i = \sum_{j \neq y_i} [\max(0, w_j^T x_i - w_{y_i}^T x_i) + 1]$$

式から明らかなように，各例のデータ損失は，Wの(max(0,−)関数によるゼロ閾値の)線形関数の和である．さらに，Wの各行(すなわち，W_j)の前に正の符号を持つこともあれば (例題の間違ったクラスに対応する場合)，負の符号を持つこともある (例題の正しいクラスに対応する場合)．これをより明確にするために，3つの1次元の点と3つのクラスを含む単純なデータセットを考えてみましょう．完全なSVMの損失(正則化なし)は次のようになります．

$$L_0 = \max(0, w_1^T x_0 - w_0^T x_0 + 1) + \max(0, w_2^T x_0 - w_0^T x_0 + 1)$$

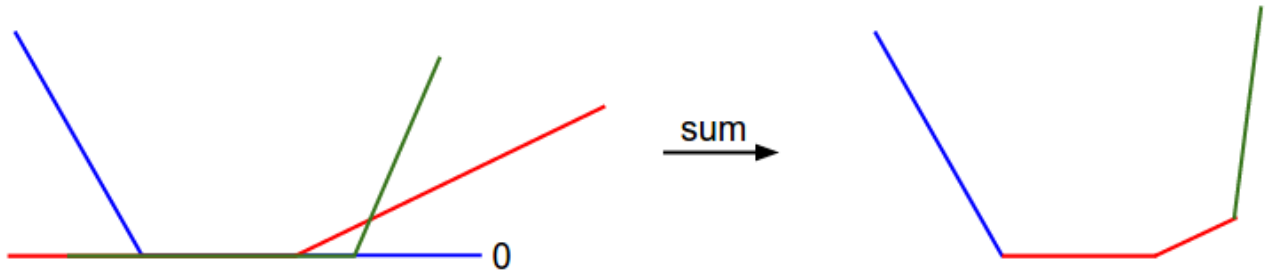
$$L_1 = \max(0, w_0^T x_1 - w_1^T x_1 + 1) + \max(0, w_2^T x_1 - w_1^T x_1 + 1)$$

$$L_2 = \max(0, w_0^T x_2 - w_2^T x_2 + 1) + \max(0, w_1^T x_2 - w_2^T x_2 + 1)$$

$$L = (L_0 + L_1 + L_2)/3$$

これらの例は 1 次元なので，データx_iと重み w_jは数値である．例えば，w_0を見てみると，上のいくつかの項はw_0の線形関数であり，それぞれがゼロでクランプされています．これを次のように可視化することができます

(次ページに続きます)



データの損失を示す1次元図．x軸は重みで，y軸は損失です．データ損失は複数の項の和であり，それぞれは特定の重みから独立しているか，またはゼロで閾値が設定されている線形関数です．完全なSVMデータ損失は，この形状の30,730次元版である．

余談ですが，お椀型の外観から，SVM のコスト関数が凸関数の一例であることがお分かりになったかもしれません．このようなタイプの関数を効率的に最小化するための文献は数多く存在します．スコア関数fをニューラルネットワークに拡張すると，目的関数は非凸型になり，上の可視化ではボウルではなく，複雑ででこぼこした地形が特徴的になります．

非微分性の損失関数．技術的な注意点として，(max演算による)損失関数のねじれが，技術的に損失関数を非微分化していることがわかります．しかし，副勾配はまだ存在し，その代わりに一般的に使われています．このクラスでは，サブグラジエント(劣勾配)とグラジエント(勾配)という用語を互換的に使います．

最適化

最適化の目標は，損失関数を最小化するWを見つけることです．ここでは，損失関数を最適化するためのアプローチを，モチベーションを高めながらゆっくりと開発していきます．この授業に参加したことがある人にとっては，このセクションは凸問題であるため，奇妙に見えるかもしれませんが，私たちの目標は，最終的には，凸最適化の文献で開発されたツールを簡単には使用できないニューラルネットワークを最適化することであることを覚えておいてください．

作戦その1．非常に悪い解決策：ランダム検索

与えられたパラメータWのセットがどれだけ良いかをチェックするのは非常に簡単なので，最初に思いつくかもしれない（非常に悪い）アイデアは，単に多くの異なるランダムな重みを試してみて，どれが最も良く機能するかを追跡することです．この手順は次のようになります．

(次ページに続きます)

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in range(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)
# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (trunctated: continues for 1000 lines)
```

上のコードでは，いくつかのランダムな重みベクトルWを試してみましたが，いくつかはうまくいきました．この検索で見つかった一番良い重みWをテストセットで試してみることができます．

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

最高のWでは約15.5%の精度が得られます．クラスを完全にランダムに推測しても10%しか得られないことを考えると，このような脳死状態ランダム検索の解決策にしては，それほど悪い結果ではありません．