CS231n Convolutional Neural Networks for Visual Recognition

4.バックプロパゲーション(誤差逆伝播法)

序論

このセクションの学習にあたって．このセクションでは，連鎖法則の再帰的適用によって式の勾配を計算する方法であるバックプロパゲーション(誤差逆伝播法)を直感的に理解できる知識を身につけます．この過程とその微妙な違いを理解することは，ニューラルネットワークを理解し，効果的に開発，設計，デバッグするために不可欠です．

問題提起．本セクションで研究する中核的な問題は以下の通り．ある関数$f(x)$が与えられ(xは入力ベクトル)，xにおけるfの勾配(すなわち$\nabla f(x)$)を計算することに重点を置きます．
学習動機．この問題が興味深い第一の理由は，ニューラルネットワークの場合，fは損失関数(L)に対応し，入力xは訓練データとニューラルネットワークの重みで構成されることです．例えば，損失はSVMの損失関数であり，入力は訓練データ$(x_i, y_i), i = 1...N$と重みおよびバイアスW, bの両方です．機械学習では通常のことですが，学習データを与えられた固定されたものとして考え，重みを制御できる変数として考えてください．したがって，バックプロパゲーションを使って入力例x_iの勾配を簡単に計算することができますが，実際には通常，パラメータ(W, bなど)の勾配を計算するだけで，パラメータの更新に使用することができます．しかし，この授業の後半で見るように，x_iに対する勾配は，例えばニューラルネットワークが何をしているかを可視化したり，解釈したりする目的で役に立つことがあります．
受講する方の中で連鎖律による勾配の導出に慣れている方でも，少なくともこのセクションには目を通していただきたいと思います．このセクションでは，バックプロパゲーションを実値回路のバックワード・フローとして捉える，あまり発展していない見解が示されており，そこから得られる洞察は，授業全体に役立つかもしれません．

勾配についての簡単な説明と解釈

より複雑な形状の方程式の表記や慣例を身につけるために，簡単なことから始めましょう．
2つの数字の単純な乗算関数$f(x, y) = xy$を考えてみましょう．どちらの入力に対しても，偏微分を導出することはただの微積分の問題です．

$$f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y, \quad \frac{\partial f}{\partial y} = x$$

解釈．微分の意味を考えてみましょう．微分とは，ある特定の点に近い無限小の領域間の，その変数に対する関数の変化率を示すものです．

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

技術的な注意点として，左辺の分数は，右辺の分数とは異なり，除算の意味ではありません．この表記は，関数fに対して演算子$\frac{\partial}{\partial x}$が適用され，別の関数(導関数)を返すことを示しています．
上の式を考える上で良い方法は，hが非常に小さいとき，関数は直線で近似され導関数がその傾きであるという点です．例えば，$x=4, y=-3$であれば，$f(x,y)=-12$となり，xで偏微分すると$\frac{\partial f}{\partial x}=y=-3$となります．これは，変数の値をわずかに増やしても，式全体では(負の符号のために)減少し，その3倍の値になることを示しています．これは，上の式($f(x+h)=f(x)+h\frac{df(x)}{dx}$)を並べ替えるとわかります．同様に，$\frac{\partial f}{\partial x}=4$であるから，$y$の値をある非常に小さな量$h$だけ増加させれば，関数の出力も(正の符号のために)$4h$だけ増加すると予想されます．

☞各変数の微分は，その値に対する式全体の感度を教えてくれます．

前述のように，勾配∇fは偏導関数のベクトルなので，$\nabla f=[\frac{\partial}{\partial x},\frac{\partial}{\partial y}]=[y,x]$となります．勾配は技術的にはベクトルですが，簡単にするために，技術的に正しい表現である「x上の偏導関数」の代わりに「x上の勾配」などの用語を使うことがあります．また，足し算の導関数を導くこともできます．
また，足し算の導関数を導くこともできます．

$$f(x,y)=x+y \quad \rightarrow \quad \frac{\partial f}{\partial x}=1 \quad \frac{\partial f}{\partial y}=1$$

つまり，x,yの両方での微分は，x,yの値に関わらず1となります．これは，x,yのいずれかを増加させることでfの出力が増加し，その増加率はx,yの実際の値に関係しないので，理にかなっています(上記の乗算の場合とは異なります)．
この授業でよく使う最後の関数は，max演算です．

$$f(x,y)=\max(x,y) \quad \rightarrow \quad \frac{\partial f}{\partial x}=\mathbb{1}(x>=y) \quad \frac{\partial f}{\partial y}=\mathbb{1}(y>=x)$$

つまり，(劣)勾配は大きくなった方の入力では1，もう一方の入力では0となります．直感的には，入力が$x=4, y=2$であれば，最大値は4であり，この関数はyの設定には敏感ではありません．つまり，もしわずかにhを増加させたとしても，関数は4を出力し続け，したがって勾配はゼロになります．もちろん，yを大きく(例えば2よりも大きく)変化させれば，fの値も変化しますが，導関数は，そのような大きな変化が関数の入力に与える影響については何も教えてくれません．微分は，$\lim_{h\to 0}$と定義されていることからもわかるように入力に対する極小，無限小の変化に対してのみ情報を提供します．

連鎖律による複合式

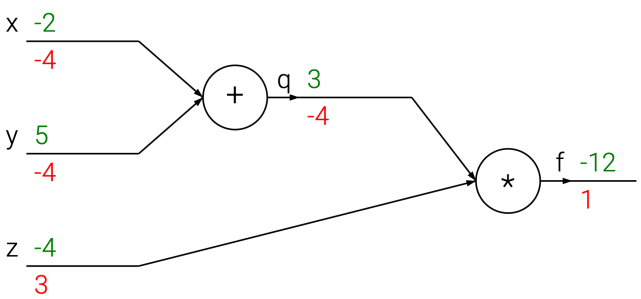
ここからは，$f(x,y,z)=(x+y)z$のような複数の関数を含む，より複雑な式を考えてみましょう．この式は，直接微分できるほど単純なものですが，バックプロパゲーションの直感を理解するのに役立つ特別なアプローチをとってみましょう．特に，この式が次の2式に分解できることに注意してください．$q=x+y$，$f=qz$．さらに，前節で見たように，両方の式の微分を別々に計算する方法もわかっています．fはqとzの乗算だけなので$\frac{\partial f}{\partial q}=z, \frac{\partial f}{\partial z}=q$，$q$はxとyの加算なので$\frac{\partial q}{\partial x}=1, \frac{\partial q}{\partial y}=1$となります．しかし，中間値$q$での勾配は必ずしも配慮することはない…$\frac{\partial f}{\partial q}$の値は役に立たないからです．その代わり，最終的に注目したいのはfのx,y,zに対する勾配です．連鎖律では，これらの勾配式を「連鎖」させるには，乗算を行うのが正しい方法です．例えば，$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$です．
実用的には，2つの勾配を持つ2つの数値を掛け合わせるだけです．例を見てみましょう．

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

変数`[dfdx,dfdy,dfdz]`の勾配が残り，変数`x,y,z`の`f`に対する感度を知ることができます．これがバックプロパゲーションの最も単純な例です．今後は，`df`という接頭辞を省いた，より簡潔な表記法を使うことにします．例えば，`dfdq`ではなく`dq`と書き，常に最終出力で勾配が計算されていると仮定します．
この計算は，神経回路図を使ってきれいに視覚化することもできます．



左側の実数値の「回路」は，計算を視覚的に表したものです．前進パスでは，入力から出力までの値を計算します(緑色で表示)．後方パスでは，入力から出力に至るまでの勾配(赤で表示)を計算するために，最後から始まって連鎖法則を再帰的に適用するバックプロパゲーションを実行します．勾配は，回路を逆に流れると考えることができます．

バックプロパゲーションの直観的な理解

バックプロパゲーションはとても局所的なプロセスであることに注目してください．回路図のすべてのゲートは，いくつかの入力を得て，すぐに2つのことを計算することができます．1.その出力値，2.入力に対する出力の局所的な勾配です．ゲートは，自身が組み込まれている回路全体の詳細を意識することなく，完全に独立してこの処理を行うことができることに注目してください．しかし，フォワードパスが終わると，バックプロパゲーションの際に，ゲートは最終的に回路全体の最終出力に対する出力値の勾配を知ることになる．チェーンルールでは，ゲートはその勾配を，通常，すべての入力に対して計算するすべての勾配に乗じるべきだとしている．

☞この連鎖法則による余分な乗算(各入力に対して)は，単一の比較的役に立たないゲートを，ニューラルネットワーク全体のような複雑な回路の「歯車」に変えてしまいます．

これがどのように機能するか，もう一度例を見て直感的に理解してみましょう．加算ゲートは入力[-2, 5]を受け取り，出力3を計算しました．このゲートは加算演算を行っているため，両方の入力に対する局所勾配は+1となります．回路の残りの部分では，最終的な値として-12が計算されます．チェーンルールが回路を再帰的に遡って適用されるバックワードパスの間に，(乗算ゲートの入力である)加算ゲートは，その出力の勾配が-4であったことを学習します．回路を「より高い値を出力したい」と擬人化すると(これは直感を助けることになる)，回路は加算ゲートの出力を(負の符号のために)より低く，力を4にすることを「望んでいる」と考えることができる． 再帰を続けて勾配を連鎖させるために，加算ゲートはその勾配を取り，入力のローカルな勾配すべてに乗算する(xとyの両方の勾配を1 * -4 = -4にする)
これが望ましい効果であることに注目してください：x, yが減少すると(負の勾配に反応して)，加算ゲートの出力が減少し，その結果，乗算ゲートの出力が増加します．つまりバックプロパゲーションとは，最終的な出力値を高くするために，ゲートがお互いに(勾配信号を通して)出力の増減(およびその強さ)を伝え合っていると考えることができます．

モジュラリティ(分割することによる効果)：シグモイドの例

先に紹介したゲートは比較的自由に設定することができます．微分可能な関数であればどのようなものでもゲートとして働くことができますし，複数のゲートを1つのゲートにまとめたり，関数を複数のゲートに分解したりすることも都合の良いときにできます．この点を説明するために，別の式を見てみましょう．

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

後述するように，この式はシグモイド活性化関数を使用する2次元のニューロン(入力xと重みwを持つ)を表しています．しかし今は，入力w, xから1つの数値への関数として，非常にシンプルに考えることができます．この関数は複数のゲートで構成されています．上で説明したゲート(add, mul, max)に加えて，4つのゲートがあります．

$$
\begin{aligned}
f(x) &= \frac{1}{x} && \rightarrow && \frac{df}{dx} = -1/x^2 \\
f_c(x) &= c + x && \rightarrow && \frac{df}{dx} = 1 \\
f(x) &= e^x && \rightarrow && \frac{df}{dx} = e^x \\
f_a(x) &= a\,x && \rightarrow && \frac{df}{dx} = a
\end{aligned}
$$

ここで，関数f_c, f_aは，それぞれ入力を定数cで変換し，定数aでスケーリングします．これらは技術的には加算と乗算の特殊なケースですが，定数c, aの勾配が不要なので，ここでは(新しい)単項ゲートとして導入します．回路全体の構成は次のようになります．