自然言語処理
―ニューラルネット―

https://satoyoshiharu.github.io/nlp/

100本ノック第7章と第8章の位置づけ

- １００本ノック課題集では、第7章が単語ベクトルで、第8章がニューラルネットとなっています。第7章の課題は、単語ベクトルがあるとしてこれを利用するだけなので、その順序になっています。
- しかし、単語ベクトルは、ニューラルネットを使って構築するものなので、学習する論理的な順序としては逆です。論理的な順序で第8章->第7章と進むことをお勧めします。
- 第7章には、機械学習のトピックであるクラスタリングと次元圧縮が含まれます。そのため、第7章の課題を済ませて機械学習をカバーしてから、第8章のニューラルネットの課題に取り組むのでも、OKです。

以下、ニューラルネットの基本的な部分の概念を説明します。ニューラルネットの入門書一冊に相当します。それをご自分のペースで咀嚼した後で、課題に取り組んでください。

自然言語処理：ニューラルネット
Deep Learning はじめの一歩

解説動画

ニューラルネット、Deep Learningという技術潮流に関して、その最初の一歩として、なぜこれが興隆しているかを説明します。

神経細胞

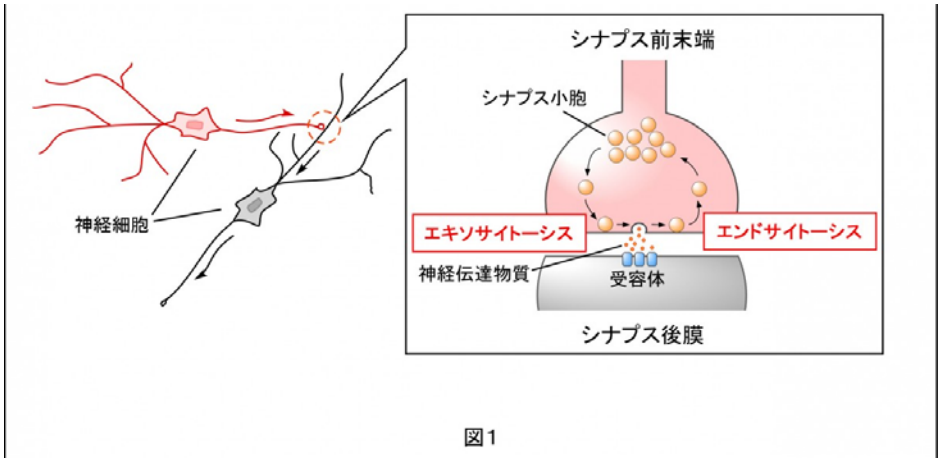
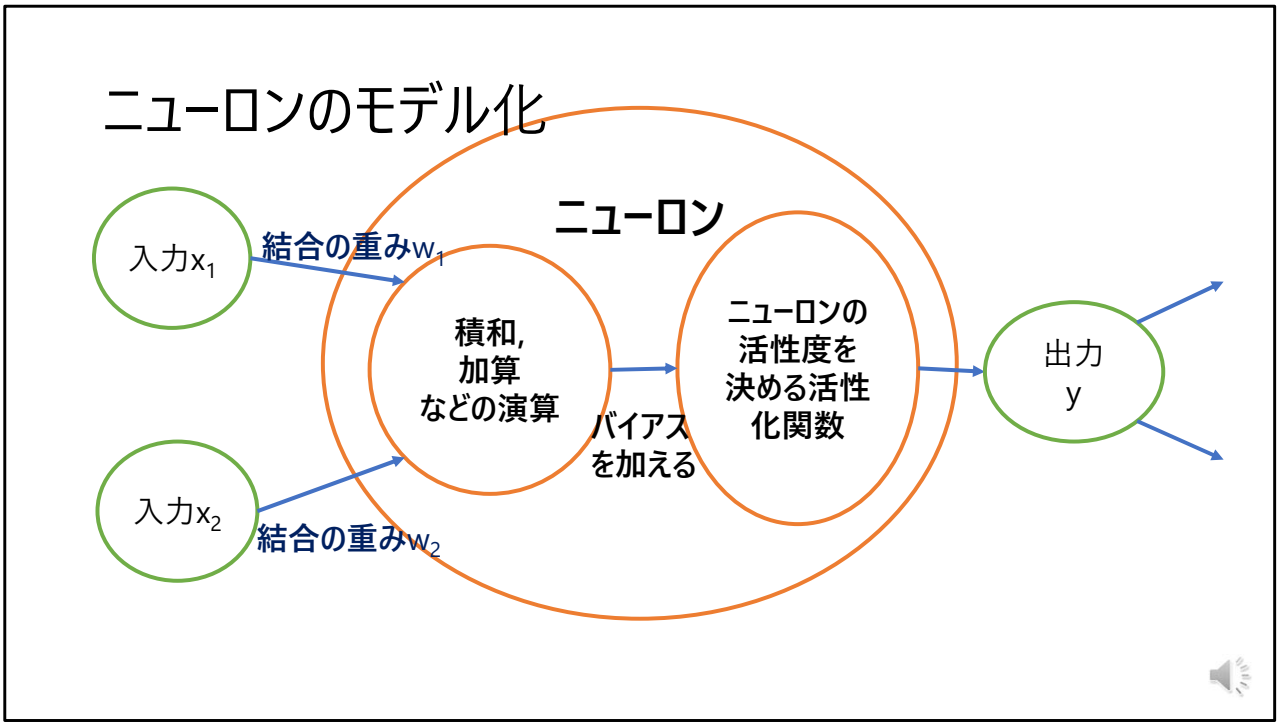


図1

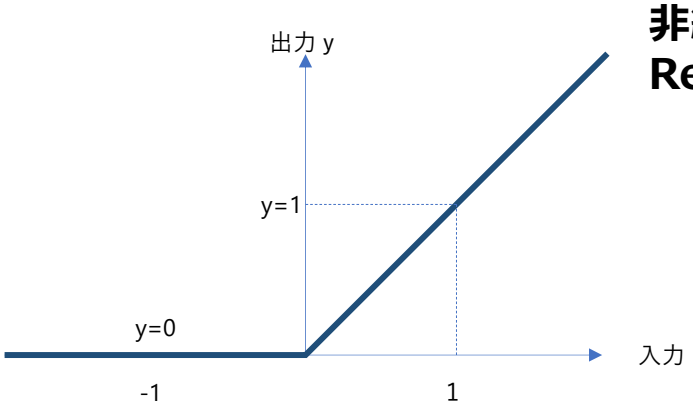
一つの神経細胞内では、情報が電流として流れます。
ある神経細胞はほかの神経細胞と、伝達物質を通してつながっています。



ニューラルネットは、人の神経をまねたニューロンを構成要素としています。
ニューロンには、ほかのニューロンからきた入力データと、ほかのニューロンに伝える出力データがあります。
神経細胞の中では、入力側のニューロンの興奮度合い対して、なにがしかの演算をして、自分の興奮度合いを決め、ほかのニューロンに情報を伝えます。
ニューラルネットを作るときは、このなにがしかの演算処理や活性度を決める処理を、なんとか層という言い方をします。
なにがしかの演算のところは、行列の積だったり、畳み込みだったり、加算だったり、応用に応じて設定します。

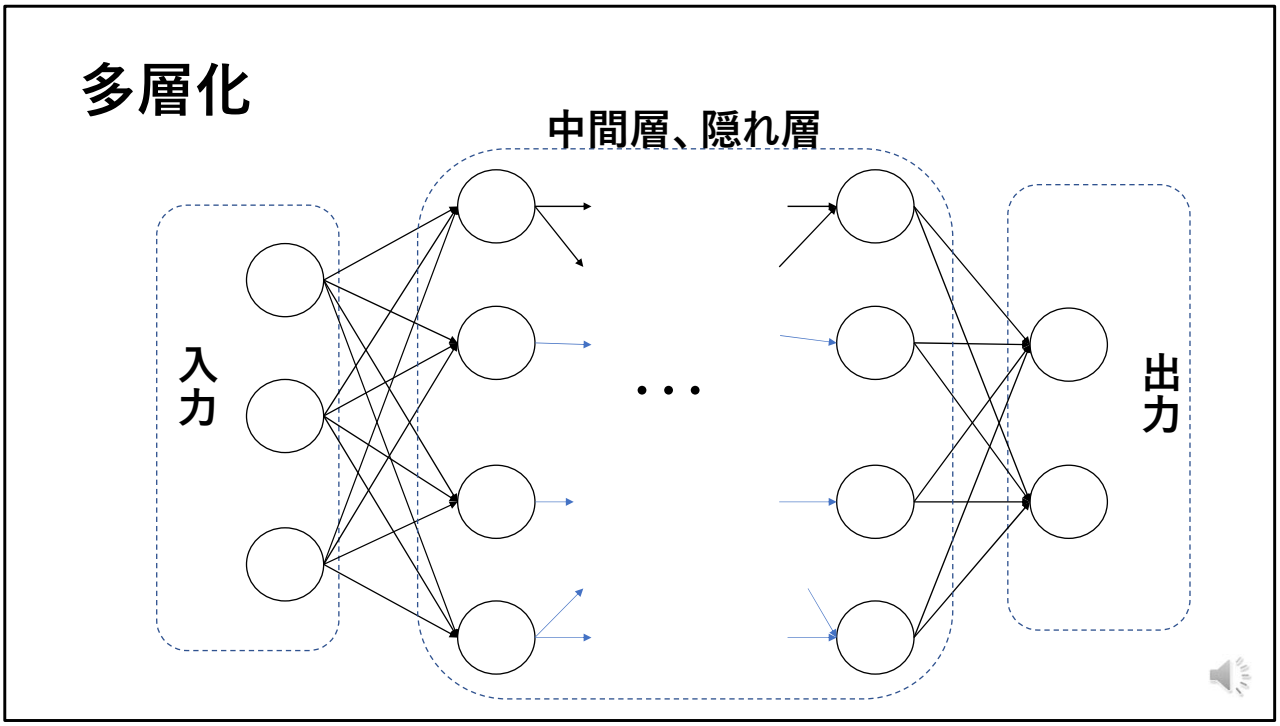
活性化(Activation)関数

非線形変換
ReLU



線形関数とは、f(x+y)=f(x)+f(y), f(ax)=af(x) であるような関数。
ReLUは**非**線形関数: ReLU(-1+1) = 0, ReLU(-1) + ReLU(1) = 0 + 1 = 1

活性度を決める層では、非線形関数が使われます。代表的なものは、ReLUです。
線形関数というのは、足し算と掛け算、四則演算と組み合わせても、予測しやすい、素直な、ストレートな対応関係です。
グラフが直線になる一次式などが、線形関数です。
一方、非線形関数というのは、線形でない対応です。
つまり、四則演算と組み合わても、素直な、ストレートな対応関係がない、素直でない、複雑な対応関係です。



Deep Learningは、従来のニューラルネットに比べて、層を何重も重ねることが特徴です。
層の中には非線形対応が組み込まれているので、非線形対応が何個も重ねられていることになります。

線形変換は、何層繰り返しても、1層で表現できるという性質を持っています。
そのため、非線形変換をかましているがゆえに、多層化が効果を発揮できます。

Deep Learning は、入力に対する**非**線形変換を**何層**も繰り返して、複雑な入出力関係を表現する。

その結果、複雑なIO対応関係の表現力を持ちました。
非線形変換を利用すること、何層も重ねること、これが表現力の秘密です。

参考

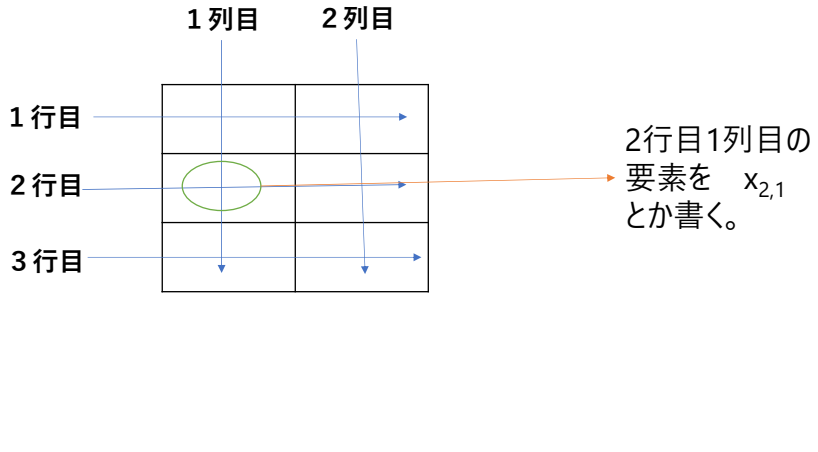
- 「ゼロから作るDeep Learning」 by 斉藤こうき、第2章、第3章
- 「Deep　Learningとは」 by Sony 小林
- 内職が要らないくらい分かりやすいディープラーニング入門
- 線形変換（一時結合）は、何層重ねても、1 個の線形変換と同じ。 -> 【大学数学】線形代数入門③(一次変換と演算の性質)【線形代数】　など
- 関数を重ねる効果 -> 【深層学習】関数 - なぜ「深さ」が AI を生み出しているのか？
- 活性化関数の選び方　->　「様々な活性化関数」 by Sony 小林

自然言語処理：ニューラルネット
全結合層
(別名：Affine層、リニア層)

解説動画

ここでは、全結合層について解説します。
全結合層は、Affine層、リニア層とも呼ばれます。

行列



最初に行列を復習します。
行列は2次元で、横方向が行、縦方向が列です。
行列の要素は、行番号・列番号のペアで指定します。

行列の積

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

行列の積を復習します。
左辺に、行列が二つ。これらをかけて、右辺を求めます。
それには、左辺の左側の行列の行と右側の行列の列をひとつづつ取り上げて、
右辺の対応する行番号、列番号の値を決めていきます。

行列の積

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

行列の積

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

行列の積

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

行列の積

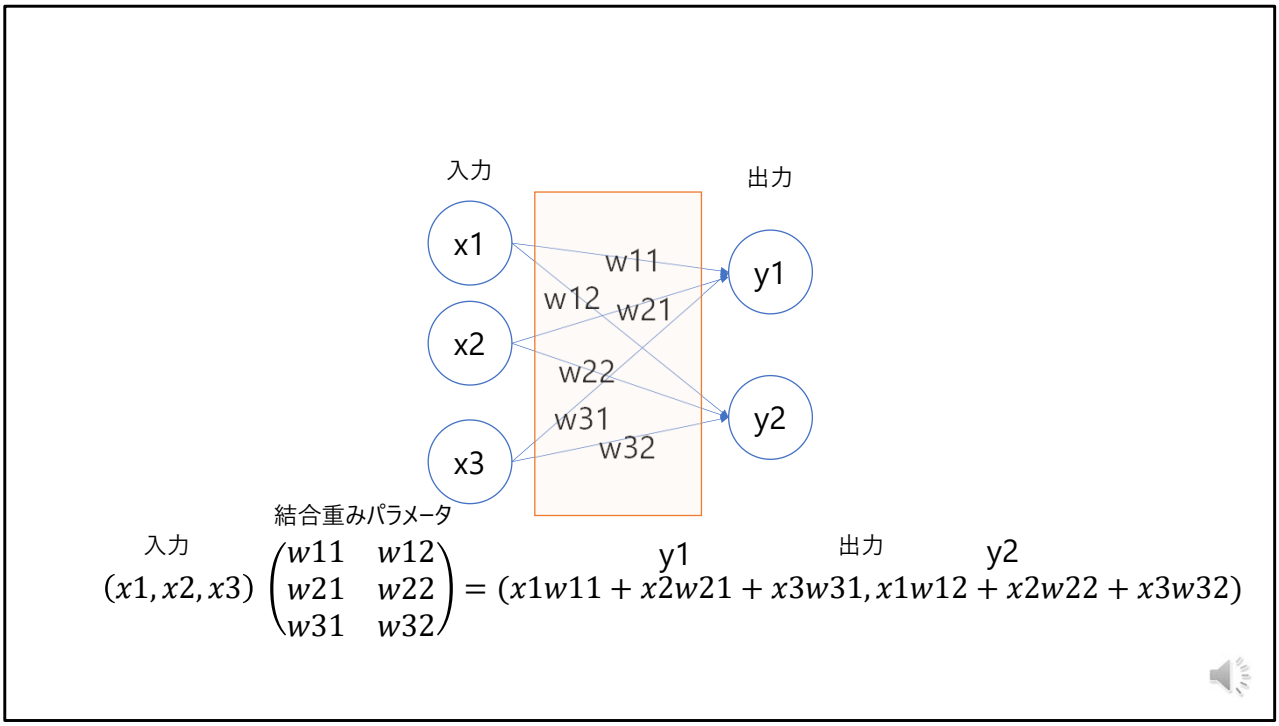
$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

行列の積

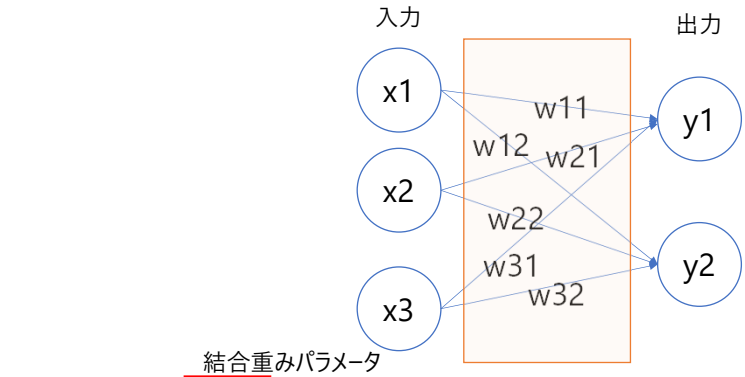
$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

行列の積

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix} \begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$



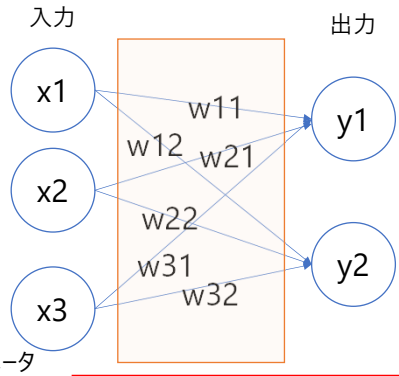
全結合層というのは、入力を出力に変換するのに、行列の積を求める処理を実行します。
入力が3個、出力が2個だとしましょう。
３ｘ２の組み合わせの結合線を考えます。それぞれの結合線には重みがあって、その重みを行列で表します。
入力がx1,x2,x3、結合重みがw11,w12,…だとします。
これら行列の積を計算することで、2個の出力y1,y2が決まります。



結合重みパラメータ

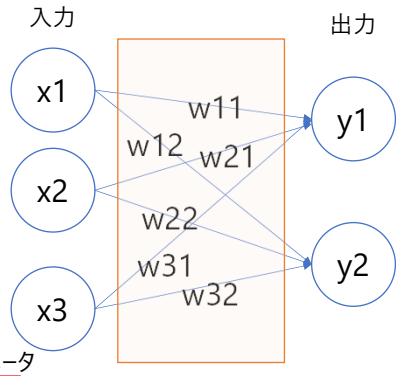
$$(x1, x2, x3)\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = (x1w11 + x2w21 + x3w31,\ x1w12 + x2w22 + x3w32)$$

y1　　y2



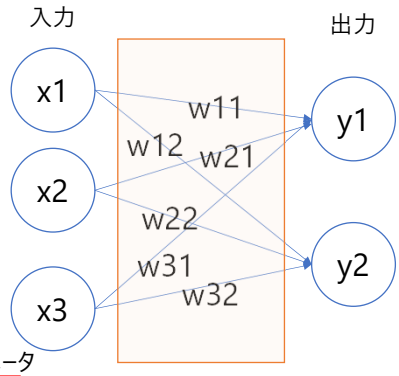
結合重みパラメータ

$$(x1, x2, x3)\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = (\underset{y1}{x1w11 + x2w21 + x3w31}, \underset{y2}{x1w12 + x2w22 + x3w32})$$



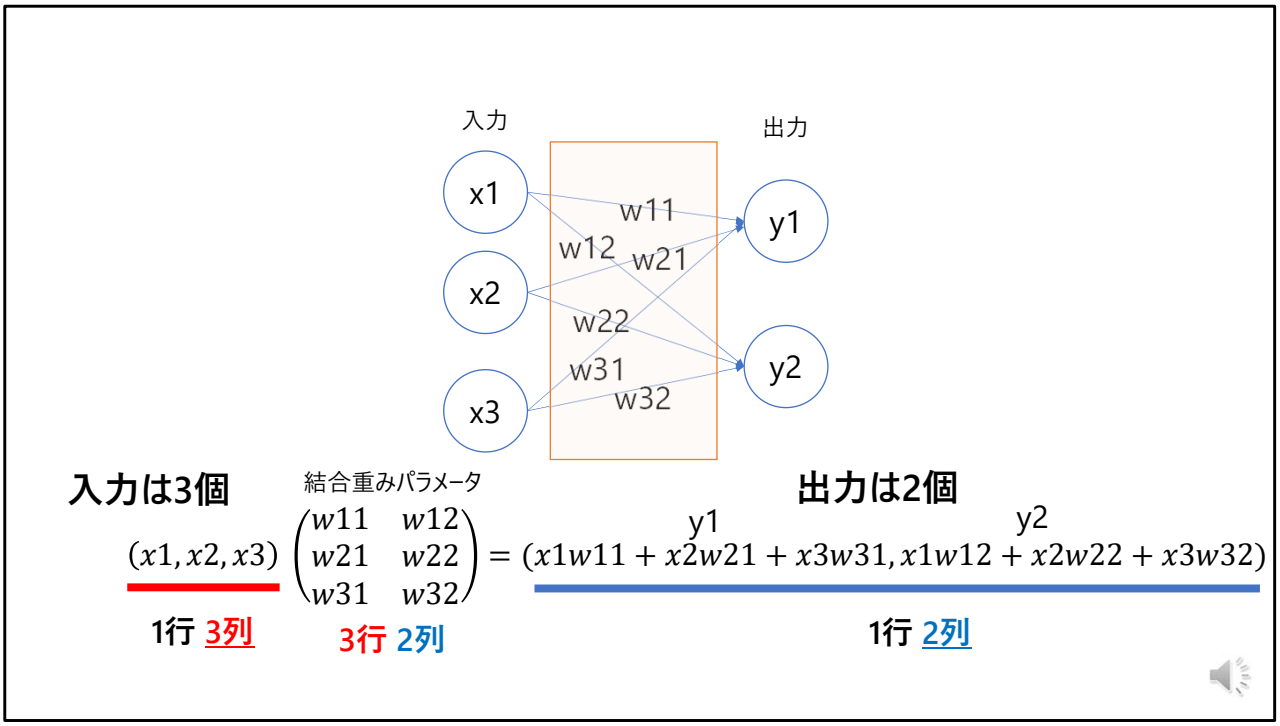
結合重みパラメータ

$$(x1, x2, x3)\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{matrix} y1 & y2 \\ (x1w11 + x2w21 + x3w31, & x1w12 + x2w22 + x3w32) \end{matrix}$$

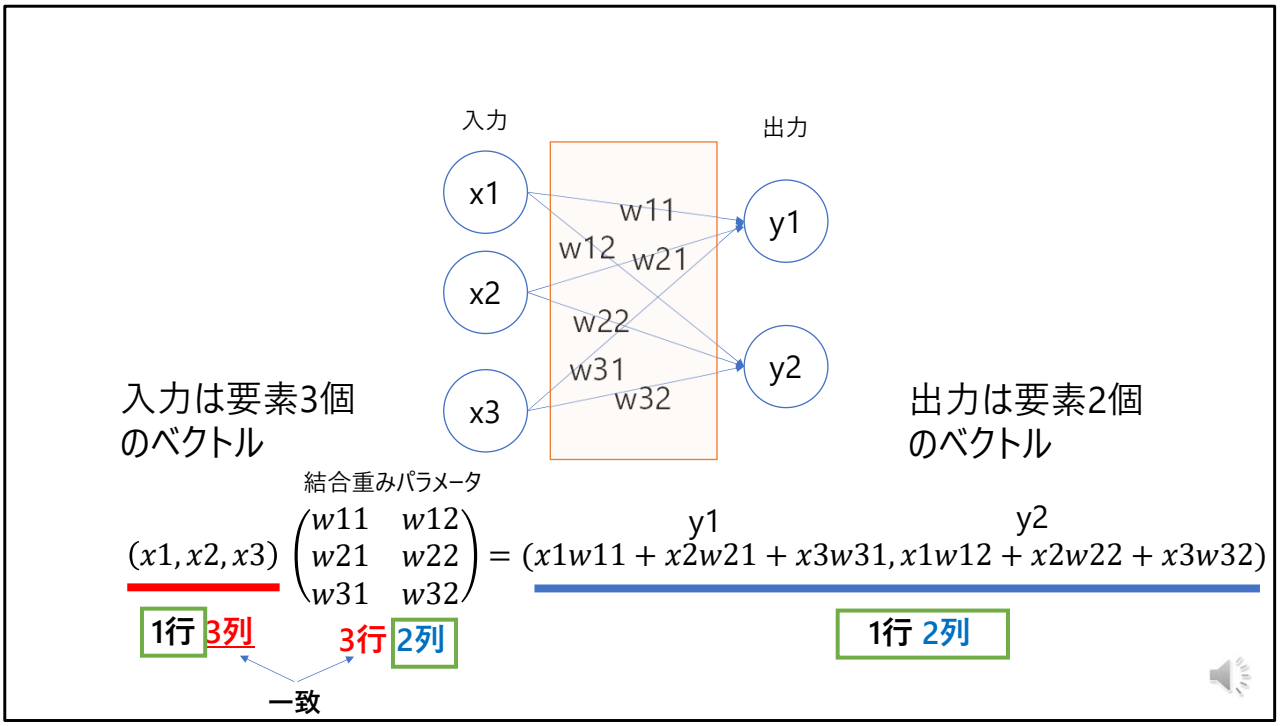


結合重みパラメータ

$$(x1, x2, x3)\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = (\underset{y1}{x1w11 + x2w21 + x3w31}, \underset{y2}{x1w12 + x2w22 + x3w32})$$

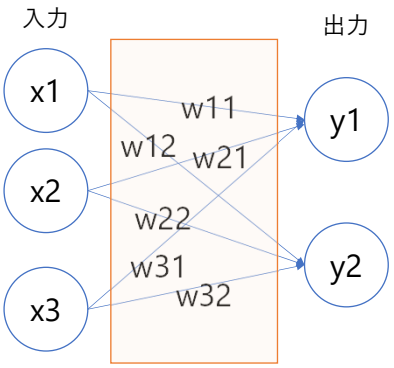


ここで、1行3列の行列と3行2列の行列をかけて、１行2列の行列を得ています。全結合層の入力が3個で出力が２個であることは、1行3列が1行２列になることに対応します。



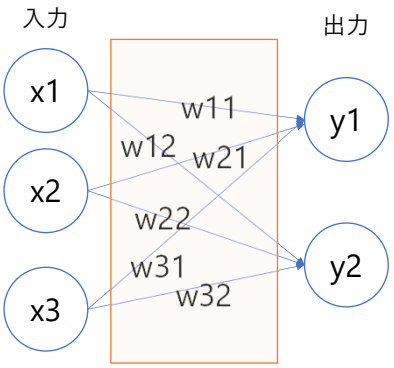
行列の積をとるとき、左側の列数と、右側の行数が一致することに、注意してください。
ニューラルネットのコードを書くときに、ここに注意しないと、要素の個数が合わないというエラーになります。

全結合層は、結合の重み行列によって、入力から何らかの特徴を取り出す



全結合層は、このような結合の重みパラメータを利用して、入力から何らかの情報変換をして特徴を出力します。

結合の重みを、正解が得られるように学習することで、望ましい特徴抽出をさせる



この結合の重みは、ニューラルネットのパラメータとして、学習時に、望ましい特徴が出力できるように最適化されます。

ミニバッチ：Training/Test最小単位

$$(x1, x2, x3)\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = (x1w11 + x2w21 + x3w31, x1w12 + x2w22 + x3w32)$$

データを何個分かまとめる

ミニバッチ
３個

$$\begin{pmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{pmatrix}\begin{pmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{pmatrix} = \begin{pmatrix} x11w11 + x12w21 + x13w31 & x11w12 + x12w22 + x13w32 \\ x21w11 + x22w21 + x23w31 & x21w12 + x22w22 + x23w32 \\ x31w11 + x32w21 + x33w31 & x31w12 + x32w22 + x33w32 \end{pmatrix}$$

3個分の
3要素の入力

3個を2個へ変換する結合重み

3個分の
２要素の出力

GPUの能力活用

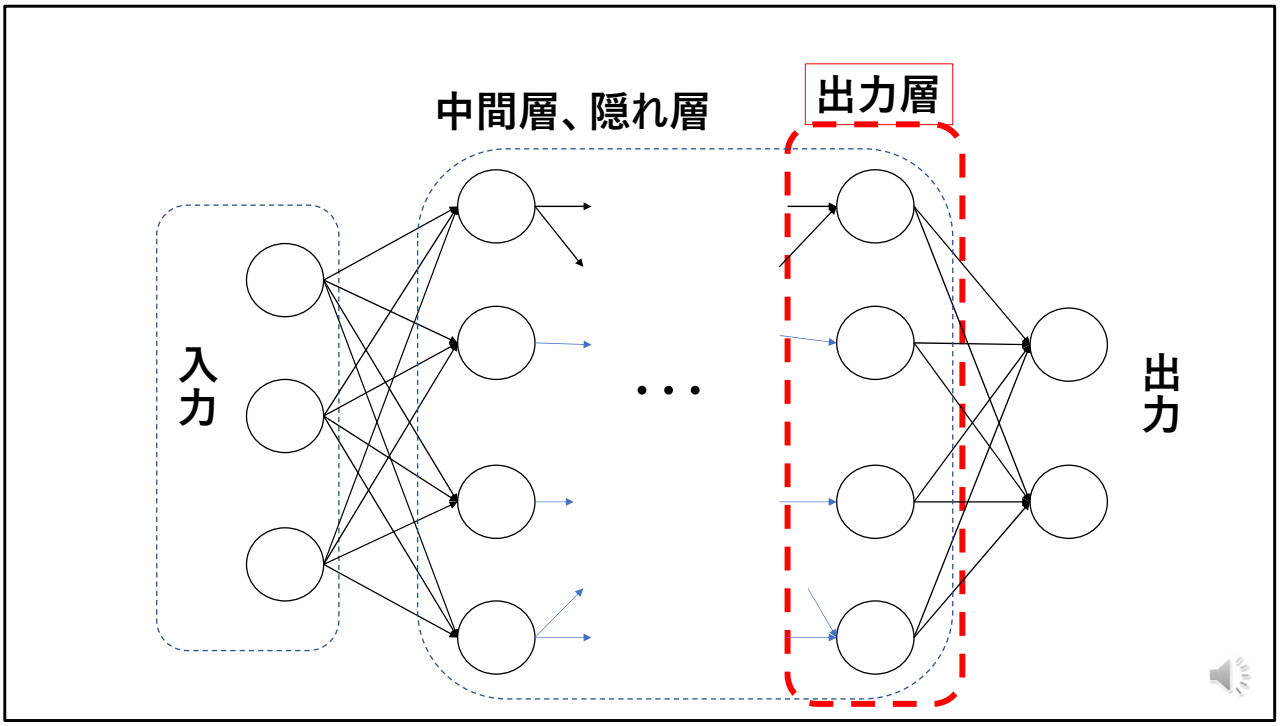
なお、ニューラルネットへの入出力は、スピードのために、複数のデータをまとめて処理するのが普通です。
そのまとまりをミニバッチといいます。
全結合の場合、ミニバッチにしても、ネットワークパラメータである重み行列は変わらず、演算処理は行列の積であり、それによって特徴抽出をするという点は変わりません。

自然言語処理：ニューラルネット

Softmax出力層

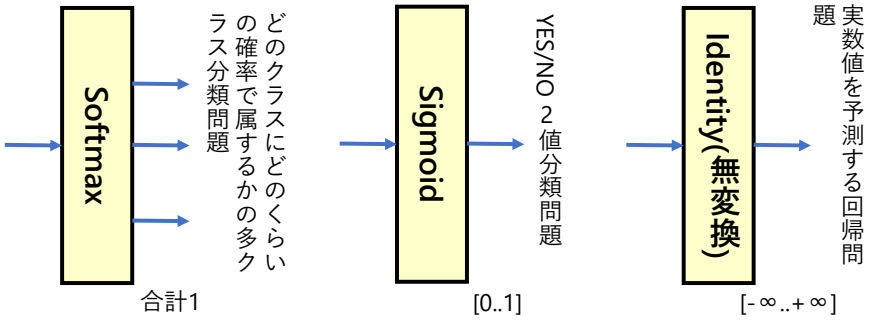
解説動画

ここでは、ニューラルネットの出力層の代表であるSoftmaxについて学びます。



ニューラルネットは、入力の処理の後に、中間層、隠れ層があり、最後に出力層があります。

出力層は望ましい最終形に変換する



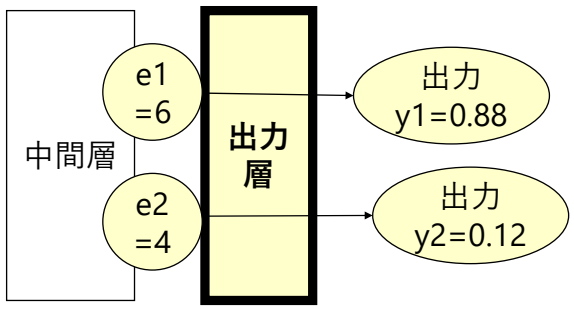
出力層の役割は、アプリの目的に応じて、望ましい出力形態に変換することです。
クラス分類問題では、出力を全部足すと1になるように変換するSoftmax関数が使われます。
クラス数分の出力ノードがあり、それぞれのクラスに属する確率が出力れていると解釈します。
どのクラスに属するかは、出力確率の最も多いノードに対応するクラスだとします。
2値問題では、0..1の範囲の値を返す、Sigmoid関数が使われます。
例えば、1に近ければTrue、0に近ければFalseなどと解釈します。
最後に、実数値を予測する回帰問題では、ネットワークの主脊力をそのまま出力します。
この場合Identity関数と呼ばれることもあります。

Softmax
合計1になるように変換する

入力(中間層の出力)

$$y_i = \frac{e^i}{\sum_j e^j}$$

出力



出力値を合計値で割れば合計 1 になるのだが、後の処理のため（逆伝播計算が楽になる）、e の出力値乗にする。

Softmax関数を詳しく見てみましょう。
中間層の出力がy1=6,y2=4だとします。
これを、足して1になるような0.88と0.12に変換します。

Softmaxの処理

$$y_i = \frac{e^{i}}{\sum_j e^{j}}$$

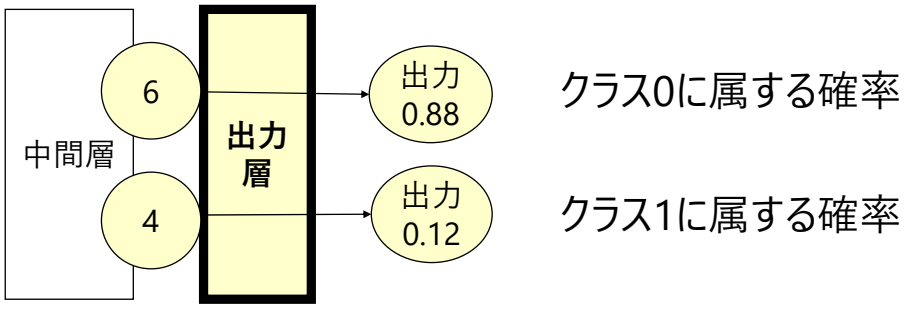
入力

出力

出力層への入力が 6 と 4 とすると、
e^6 = 403
e^4 = 55
e^6+e^4=458
なので、
$e^6/(e^6+e^4)$=0.88
$e^4/(e^6+e^4)$=0.12
これらの和は 1

オイラー数の 6 乗は403、4乗は55。それらを合わせると458
Softmaxの計算式の分母は458です。
出力が 6 ならば、6/458で、0.88
4ならば、4/458で、0.12。
0.88と0.12を合わせると 1 になります。

合計1になる数値は、あるクラスに属する確率と解釈できる



いくつかの数値が合計1の場合、これらは確率とみなすのもありです。
そう解釈してしまいましょう。
すると、出力６は、0番目のクラスに属するのが確率0.88。
出力４は、１番目のクラスに属するの確率が0.12、と解釈できます。

Softmaxは、このように、ニューラルネットの出力を、複数のクラスに属する確率へ変換することで、クラス分類問題に利用されます。

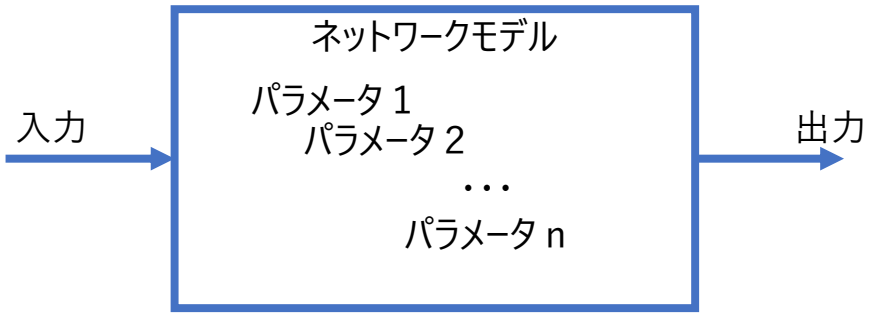
自然言語処理：ニューラルネット

クロスエントロピー・ロス関数

解説動画

ここでは、ロス関数という概念と、代表的なロス関数であるクロスエントロピーについてみていきます。

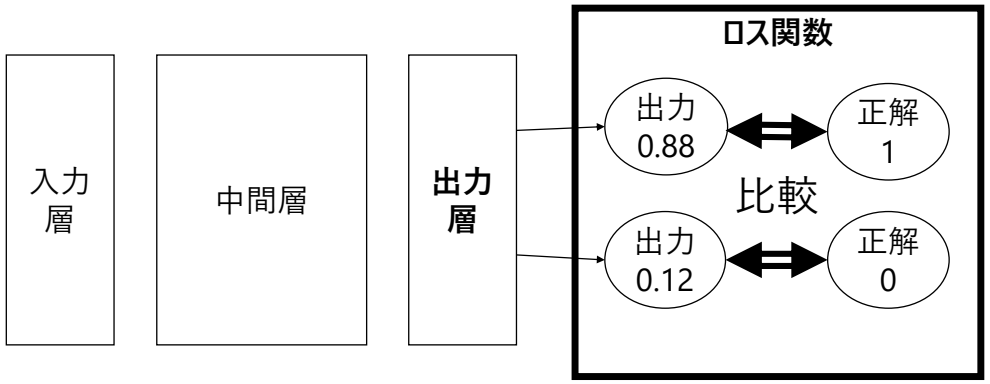
ネットワークモデルの学習とは、パラメータ（ニューロン結合の重みなど）を最適な値にすること



ニューラルネットは学習します。
学習というのは、ネットワークを構成する要素のパラメータを最適化して、
最も正解が得られやすいようにすることです。

損失（ロス）関数

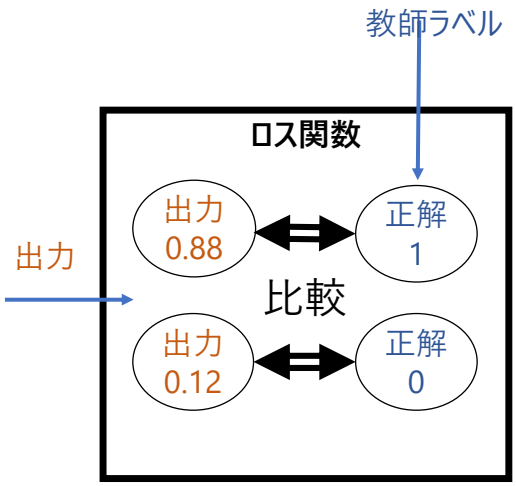
正解と、モデルの出力層の結果とを比べて、どれだけ違うかを量化する



最も正解が得られやすいようにする場合、正解がより得られているあるいはより得られていないという尺度が必要になります。
ロス関数は、その尺度を提供します。
ロス関数は、ニューラルネットが入力に対して予測した出力を、正解と比べて、どれだけロスしたか、まずかったか、をはじき出します。

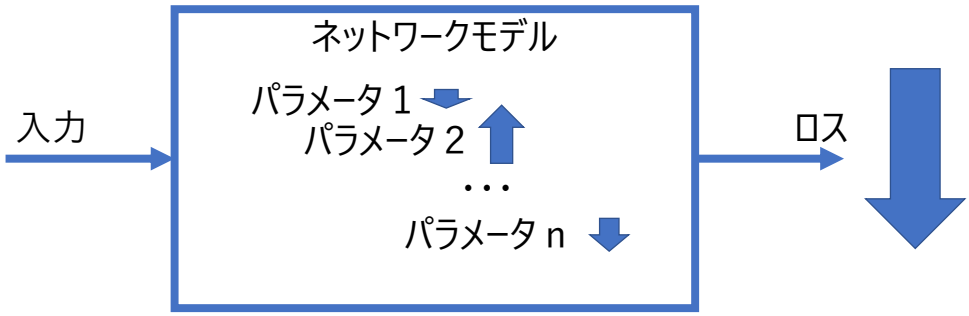
ロス関数の性質

1. 常に正である
2. 出力が正解から遠いほど大きな値になり、出力が正解に近いほど小さな値になる。



ここで、ロス関数として望ましい性質があります。
１．常に正であること。
常に正であれば、その値をできるだけ小さくするという最適化処理が起動できます。
２．出力が正解から遠いほど大きな値になり、出力が正解に近いほど小さな値になること。
出力が正解から遠いときは、ロス、まずさ加減が大きいのですから、ロス関数の値も大きくなり、逆ならば小さくなるのが望ましいです。

学習は、**ロス**（クロスエントロピー誤差など）が小さくなるように、パラメータを変更することで行う。



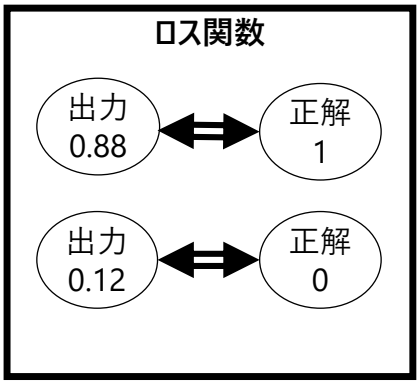
ニューラルネットの学習は、ロス関数の値がより小さくなるように、ネットワークのパラメータを最適化する処理です。
具体的には、全結合の場合、入出力ノードの結合の重み行列のそれぞれの値などです。

クロスエントロピー誤差

対応する正解ラベル

$$E = -\sum_{k} t_k \log y_k$$

k 番目の出力



代表的なロス関数である、クロスエントロピー誤差についてみていきます。
クロスエントロピー誤差は、ここに書いたような式で表されます。

対数

$$x = b^p \Leftrightarrow p = \log_b x$$

クロスエントロピーを理解するために、エントロピーというものから見ていきたいのですが、その前に、
高校数学の対数を思い出してください。

事象(確率p)の情報量 $-\log_2 p$

- コインの裏表
  - 表が出る確率pは1/2。=> 表という特定事象によって増える情報量は、$-\log_2(1/2)=-(\log_2 1-\log_2 2)=-(0-1)=1$　（$2^0=1, 2^1=2$）
  - 表裏は、１ビットで表現できる。１ビットで、裏表が特定できる。
- トランプのマーク
  - ハートが出る確率pは1/4。=> ハートという特定事象によって増える情報量は、$-\log_2(1/4)=-(\log_2 1-\log_2 4)=0+2=2$　（$2^0=1, 2^2=4$）
  - トランプのマークは、２ビットで表現できる。２ビットで、マークが特定できる。
- ルーレットを回して、東・西・南・北・南東・南西・北西・北東のいずれか
  - 確率pは、1/8。=>　東という特定事象によって増える情報量は、$-\log_2(1/8)=-(\log_2 1-\log_2 8)=0+3=3$　（$2^0=1, 2^3=8$）
  - 8方向の方角は、3ビットで表現できる。３ビットで方角が特定できる。

ある事象の確率がpであるとき、 p の、 2 を低とする対数をとって、負にしたものを情報量といいます。
例えば、コインの裏表は、
表が出る確率pは1/2です。ゆえに、そして、表という特定事象によって増える情報量は、$-\log_2(1/2)=-(\log_2 1-\log_2 2)=-(0-1)=1$です。
ところで、表裏は、１ビットで表現でき、１ビットで裏表が特定できるます。
つまり、情報量は何ビットでその事象を特定したり表現したりできるかという量です。
コインの裏表は情報量1ビット、トランプのマークは情報量２ビット。8つの方角は情報量３ビットです。
より詳しいのが、情報量が多いですね。

確率分布のエントロピー（平均情報量）

$$-\Sigma p(x)\log p(x)$$

事象 x の起きる確率　　事象 x の情報量

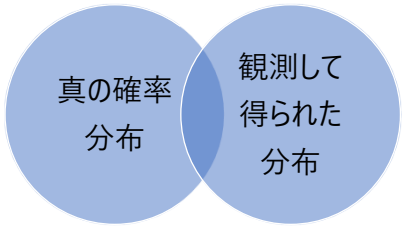
算数での平均は足して個数で割る。その代わりに、起きる確率で重みづけたものを足して、情報量の平均とする

コインの裏表の確率分布pは、p(表)＝0.5、p(裏)=0.5。一方、確率0.5の情報量は$-\log_2(1/2)=1$だった。このpのエントロピーは、-(0.5*(-1)+0.5*(-1))=1。

では、すべての事象に関する情報量の平均というものを考えます。
それぞれの事象はそれぞれ異なる確率を持っているとします。
平均を得るために、その確率で重みづけした情報量というものを考えます。
これを、平均情報量、別名エントロピーといいます。
コインの裏表の場合、そのエントロピーは１となります。
トランプのマークは、２。
方角は、３。

/*
ちなみに、エントロピー増大の法則というのを聞いたことがあると思います。
事象がどれも同じような確率で起きる場合、エントロピーは最も大きな値をとります。
事象の起きる確率がバラバラな場合、エントロピーは小さくなります。
熱いお湯と冷たい水を混ぜたとき、温度が徐々にそろっていき、エントロピーが増大します。しまいには均一な温度になってエントロピーは最大となります。
*/

クロスエントロピー（交差平均情報量）



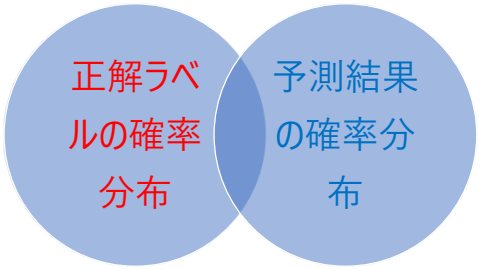
$$-\Sigma\ p(x)\log q(x)$$

真の確率で 重みづけ

x を観測して得られた情報量

エントロピーのイメージがつかめたところで、クロスエントロピーの定義を見ます。
エントロピーは、一つの確率分布に関して情報量の平均をとりました。
今度は二つの確率分布を考えます。
そして、平均をとるときの重みづけはもとのまま、一方、情報量は別の分布からとってくるとします。
これを交差平均情報量、クロスエントロピーといいます。
二つの確率分布の違いの尺度です。

クロスエントロピー



真の確率分布（重みづけ）

モデルが正しいと仮定して観測して得られた確率分布（評価対象）

$$-\Sigma\ p(x)\ \log q(x)$$

今、重みづけの確率は正解ラベルからとってきて、情報量で使う確率はニューラルネットの予測結果からとってくることを考えます。
すると、
1．クロスエントロピーは常に正であり、
2．予測結果の確率が正解と異なるほどクロスエントロピーは大きくなり、近いほど小さくなります。
これは、まさに、ロス関数として望まれる性質です。

/*
確率分布 q が、正しい確率分布 p と異なるほど大きい値になり、正しい確率分布 p と同じ時、最小値になる。

コインの裏表の確率分布pは、**p**(表)＝0.5、**p**(裏)=0.5。このpのエントロピーは、 $-\Sigma\ \mathbf{p}(x)\log\mathbf{p}(x) = -(0.5*(-1)+0.5*(-1))=1$

実際にコインを10回投げてみたら、**q**(表)=0.4、**q**(裏)=0.6となったとします。2を低とする対数の値はそれぞれ、-1.3、-0.73。

$-\Sigma\ \mathbf{p}(x)\log\mathbf{q}(x) = -(0.5*(-1.3)+0.5*(-0.73))=1.015$。

クロスエントロピーを減らす＝真の確率(正解ラベル）に近づける

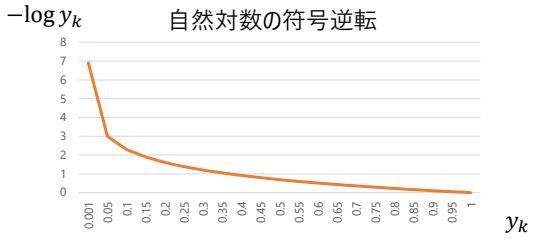
```
*/
```

クロスエントロピー誤差： 2 値の場合

対応する正解ラベル

$$E = -\sum_k t_k \log y_k$$

k番目の出力



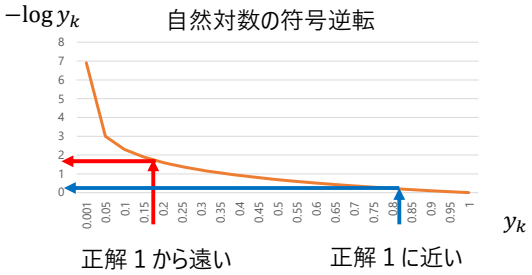
2値問題で、正解のときのラベルが 1 で、そうでないときは0とします。
すると、上の式のtkは正解の時だけ 1 で間違いの時は 0 なので、
正解の時の出力の対数を足しこむだけの計算になります。
最後に符号を逆転していることに注意。

話を単純にするために、2値問題に限定し、クロスエントロピーの性質を見ていきましょう。
正解ラベルは1か0の世界です。
式のtkは、 1 か 0 なので、実は、クロスエントロピー誤差は正解ラベルが 1 の時の予測出力値の対数を足しこむだけです。

クロスエントロピー誤差：ロス関数の性質

$$E = -\sum_{k} \log y_k$$

正解ラベルが 1 のときの出力



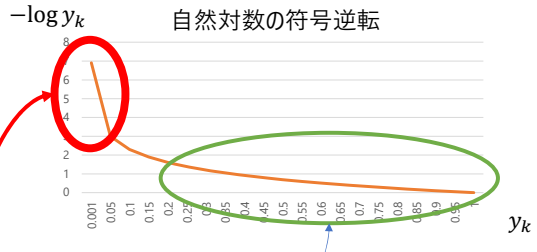
1．常に正。
2．データが正解 1 に近いほど小さな値、遠いほど大きな値。

右側の対数のグラフ（負の符号つけたもの）を見てください。
ロスEは常に正です。
そして、-logYkが、正解 1 に近いほど小さな値となり、正解から遠いほど大きな値になることが、見て取れます。

クロスエントロピー誤差：学習向き

$$E = -\sum_k \log y_k$$

正解ラベルが１のときの出力



出力が正解から遠いほど、ロスはとても大きくなり、間違えたときのペナルティが大きい。

正解に近づいたら慎重に最適点を探す

さらに、予測出力が正解から遠ければ遠いほど、その値は大きくなります。
そのため、間違えた時に、ペナルティを大きく与えているといえます。
間違えた時はたくさん学習します。
そして、そのあとは正解に近づくにつれて徐々にゆっくりと値が小さくなっていきます。
これは、慎重に慎重に最適点を探していることに対応します。
このような性質があるため、クロスエントロピーはロス関数として好まれます。

クロスエントロピー資料

- 交差エントロピーの導出
- 情報理論を視覚的に理解する
- ニューラルネットワークと深層学習、ニューラルネットワークの学習の改善

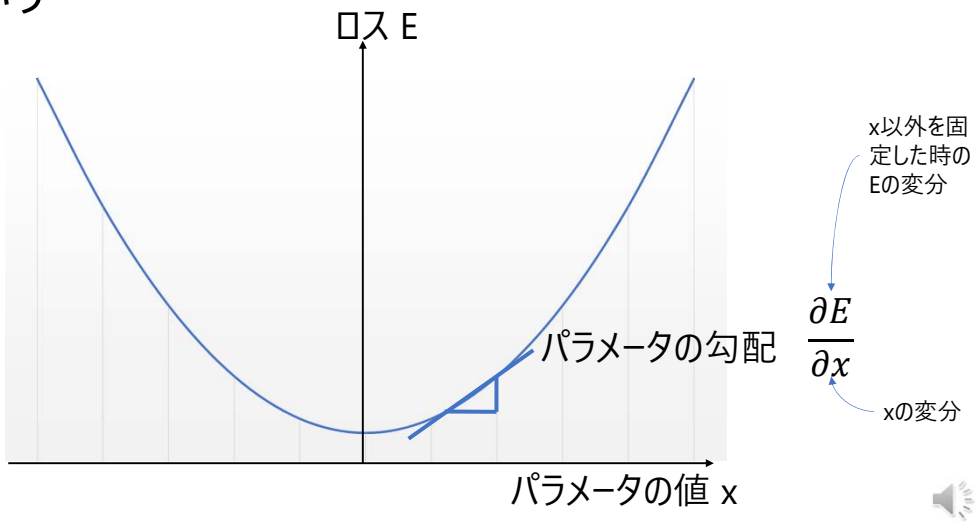
自然言語処理：ニューラルネット

逆伝播学習の仕組み

解説動画

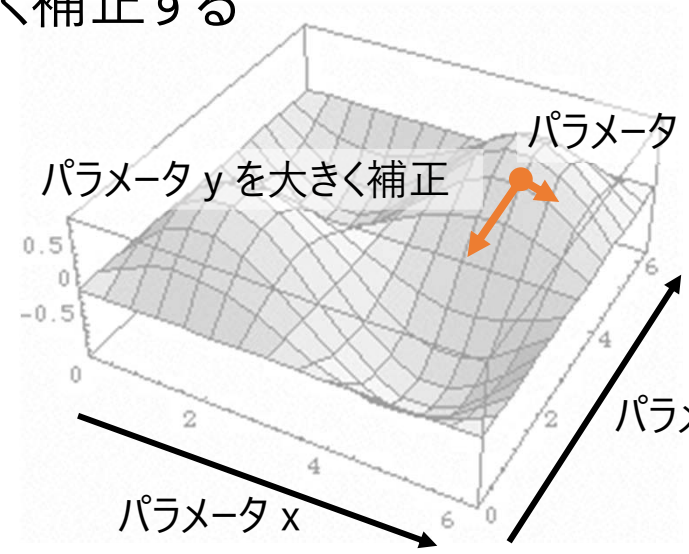
ここでは、今のDeep Learningで利用されている学習の仕組みを見ていきます。
逆伝播、バックプロパゲーションといいます。

あるパラメータをどう、どれだけ変更するか？　あるパラメータを少し変えたときにロスがどれだけ変わるかを、勾配という



ある変数xに対して変数Eの値が決まるとします。
変数 x をある値のごく近場で、わずかに動かしたとき、変数 E も動きます。
変数 E の変分を変数 x の変分で割ったものは、曲線Eの接線の傾きです。
この傾きを勾配、Gradientともいいます。

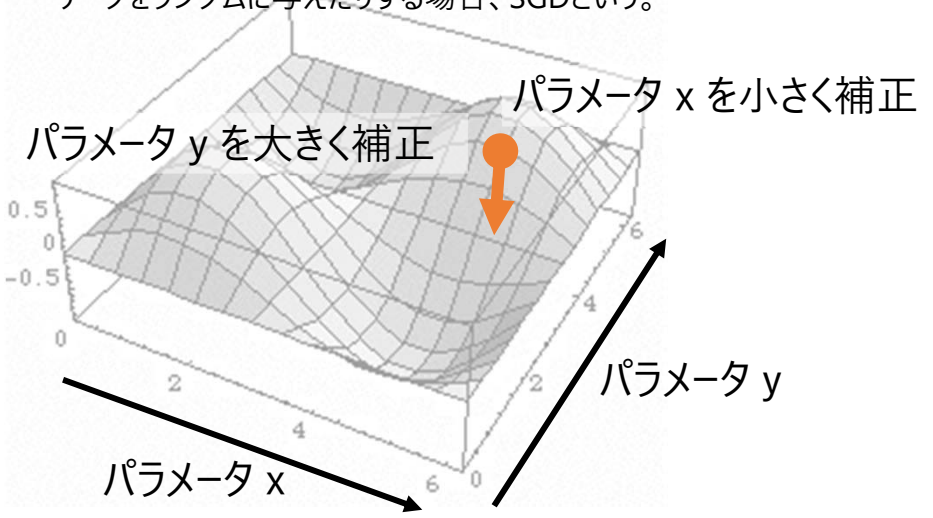
ロスが最も減るように、勾配の大きなパラメータほど強く補正する



いま、x,yの2変数がEを決めるとしましょう。
ロスEを小さくするには、ある地点から、最も下がる方向に行きたい。
そのためには、変数 x と変数 y をそれぞれ変更し、併せてもっとも下がる方向へ移動します。
例えば、変数ょりも変数y方向のほうが大きく傾いているときは、その y 方向へ大きく動き、 x の方向は小さく動きます。

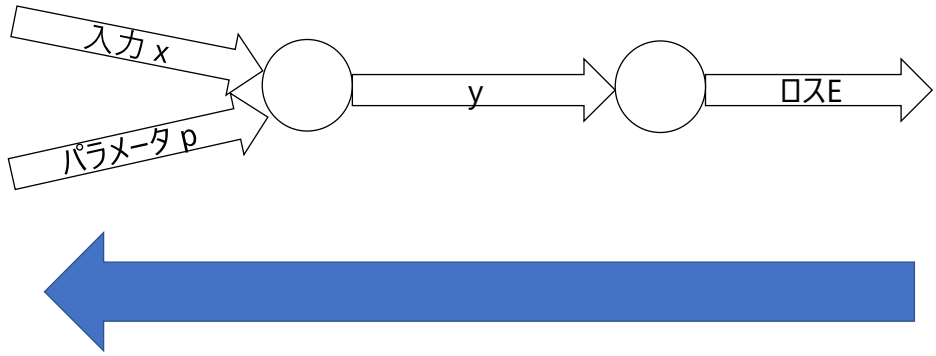
勾配降下法(Gradient Descent)

データをランダムに与えたりする場合、SGDという。



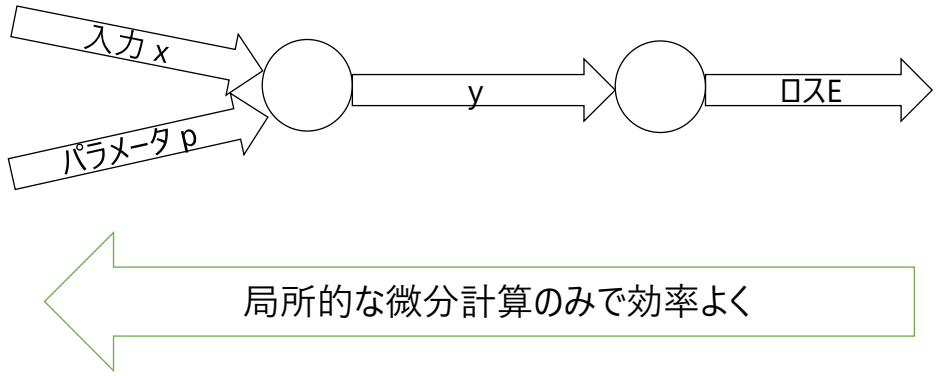
ロスEに関して、x方向、y方向それぞれの傾きに応じて、 x と y をそれぞれ補正し、最も早くEを小さくする方法を、
勾配降下法、Gradient Descentといいます。
ニューラルネットのトレーニング中、ある入力データを与えてその正解ラベルに近くなるように、
勾配降下法などによって、パラメータ x 、 y を補正することを繰り返します。
その時、データを与える順序をランダムにするなど、不確定要素があるので、
確率的勾配降法、SGD、Stochastic Gradient Descentと呼ばれます。

逆伝播：各パラメータのロスに対する勾配を、計算過程に沿って逆方向に求めていく



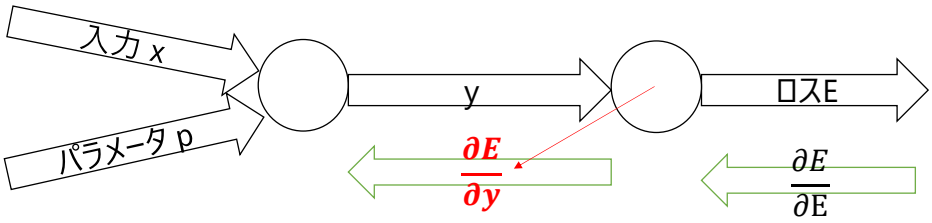
勾配降下法、SGDのイメージをつかんだところで、逆伝播、バックプロパゲーションの仕組みを見ていきます。
バックプロパゲーションは、ネットワークのそれぞれのパラメータのロスに対する勾配を、計算過程の逆順に、後ろ向きに求めるアルゴリズムです。

逆伝播：各パラメータのロスに対する勾配を、局所的な微分計算で求めていく



各パラメータの勾配を、後ろから求めていくのですが、局所的な微分計算のみで効率よく計算します。
その様子を見ていきます。

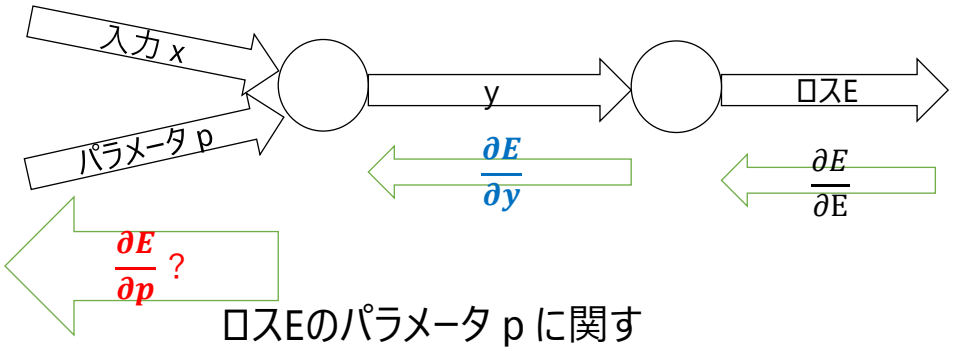
逆伝播：各パラメータの勾配を、局所的な微分計算で求めていく



ロスEの y に関する傾き

まず、ロスEを決めた最後のパラメータの一つがyだとします。
ここで、Eを y で微分し、Eのyに関する関する傾きを求めます。
これは、yを少し変えるとロスEがどのくらい変わるか、という量です。
いわば最終的なロスEに対して y がどのくらい責任あるか、あるいは貢献できるか、という量です。

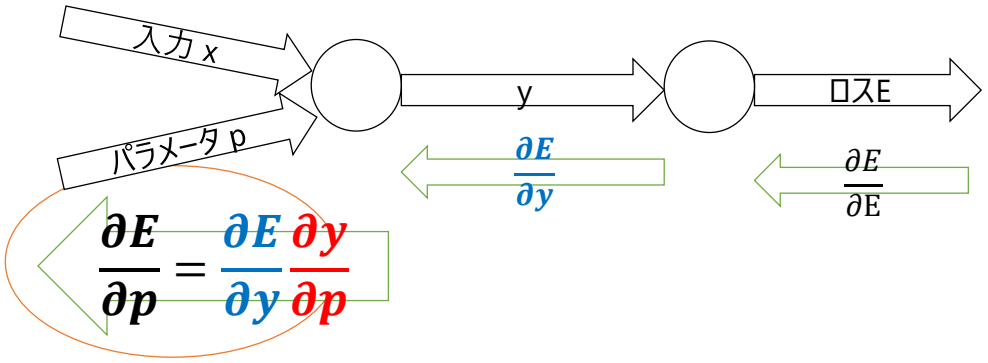
逆伝播：各パラメータの勾配を、局所的な微分計算で求めていく



ロスEのパラメータ p に関する傾き???

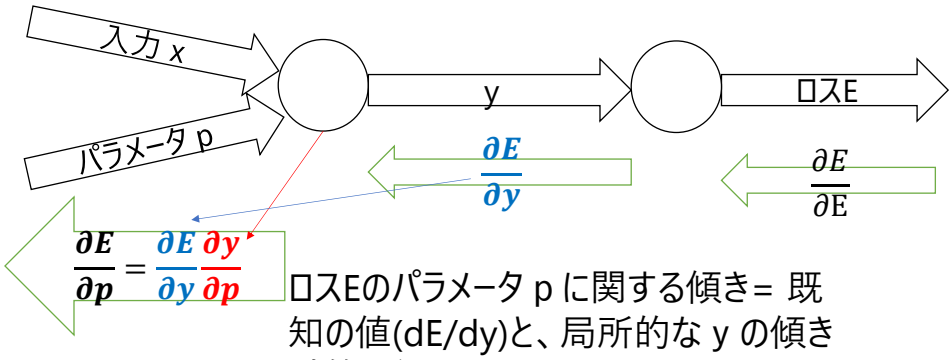
ロスEのyに関する傾きが得られました。今度は、 y を決めたパラメータの一つが p だとします。
今度は、ロスEのパラメータ p に関する傾きを求めます。
これは、 p を少し変えるとロスEがどのくらい変わるか、という量です。
いわば最終的なロスEに対して p がどのくらい責任あるか、貢献できるか、という量です

逆伝播：合成関数の微分のチェインルール



ここで、合成関数の微分のチェインルールという公式を使います。
その公式とは、E/pは、E/y＊y/p　に等しいという式です。
この公式は、右辺の左の項の分母と、右の項の分子が同じで、消去すれば、
左辺になるので、覚えやすいです。

逆伝播：各パラメータの勾配を、局所的な微分計算で求めていく

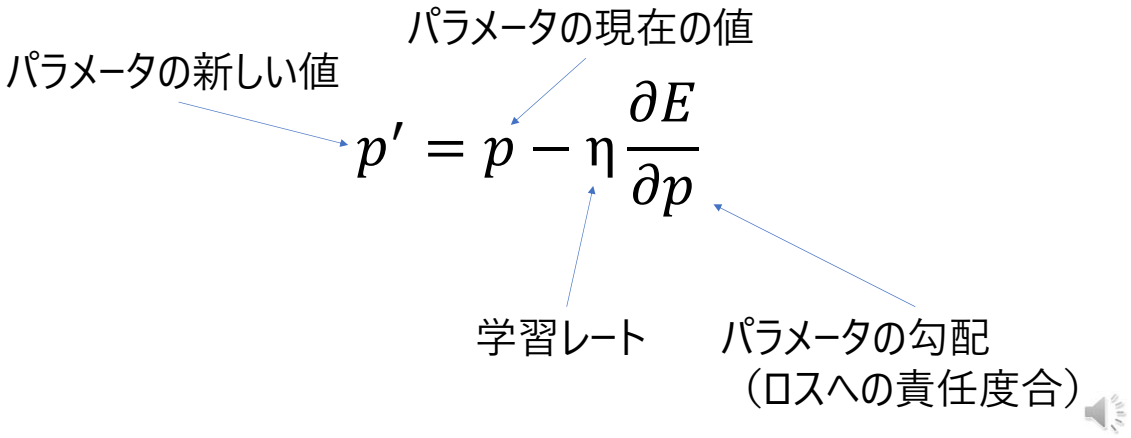


ロスEのパラメータ p に関する傾き= 既知の値(dE/dy)と、局所的な y の傾き計算で得られる

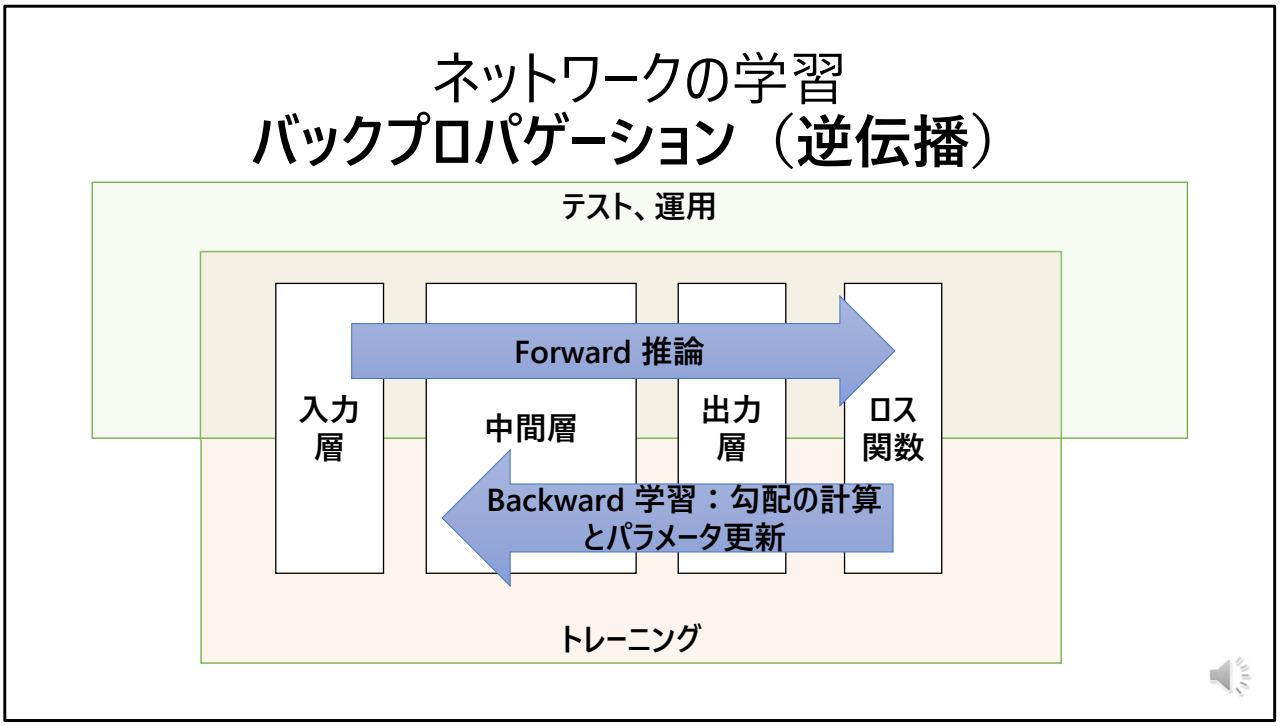
合成関数の微分のチェインルールを使うと、
左辺の、ロスEのパラメータ p に関する傾きというのは、先ほど求めた既知の値(ロスEの y に関する傾き)と y のpに関する傾きから
得られることがわかります。
y の p に関する傾きというのは、 y を計算するときに p を使ったがその時の y の計算を変数 p で偏微分したものです。
このように、後ろから計算していけば、あとは局所的な傾き計算だけで、最終的なロスEの途中のパラメータに関する傾きを
求めることができます。
これを、すべてのネットワークのパラメータに関して求めます。

（ReLUは、勾配が 1 なので、逆伝搬が伝わりやすく、学習しやすい）

各パラメータの勾配を決めたら、勾配に応じてパラメータを更新する



すべてのパラメータに関して、ロスEに対する傾きが得られました。
そしたら、各傾きを利用して、ロスEが小さくなるように（つまり、正解ラベルに近づけるように）、
勾配降下法などで、各パラメータを補正します。
この時、傾き・勾配をどの程度効かせるかを、学習レートといいます。
学習レートが大きいと、学習のスピードは速まりますが、最適でない地点に降りてしまうリスクが高まります。
学習レートが小さいとと、学習のスピードは遅くなりますが、最適点に降りないリスクが減ります。



ネットワークのモデルをトレーニングするときは、
１．入力データを与えて、前向きに推論して、予測結果を出力し、
２．予測結果と正解ラベルを比較してロスを計算し、
３．ロスに基づいて、BackPropagationをしてパラメータをロスが小さくなうように補正する、
ことを、繰り返します。
テスト及び運用時は、BackPropagationをしないで、前向き予測結果のみを利用します。

現在のフレームワークでは、このBackPropagationは、自動的に行われるので、ほとんど意識する必要はありません。
が、BackPropagationを起動する1行の意味を理解していないと、ニューラルネットのコード全体の理解ができません。

PyTorchのAutograd

PyTorchなどの最新のフレームワークは、Chainerの考えを吸収し、実行時に計算過程を記録し、それをもとにバックプロパゲーションを自動的に行うことをライブラリでサポートしている。

逆伝播参考資料

- 「ゼロから作るDeep Learning」、第4章、第5章
- ニューラルネットワークの学習の仕組み
- ニューラルネットワークと深層学習、逆伝播の仕組み
- 誤差逆伝播法を宇宙一わかりやすく解説してみる | ロボット ...
- 連鎖律の原理で、誤差を「後ろ」に伝えよう
- 高卒でもわかる機械学習 (5) 誤差逆伝播法 その1
- Deep Learning精度向上テクニック：様々な最適化手法 #1

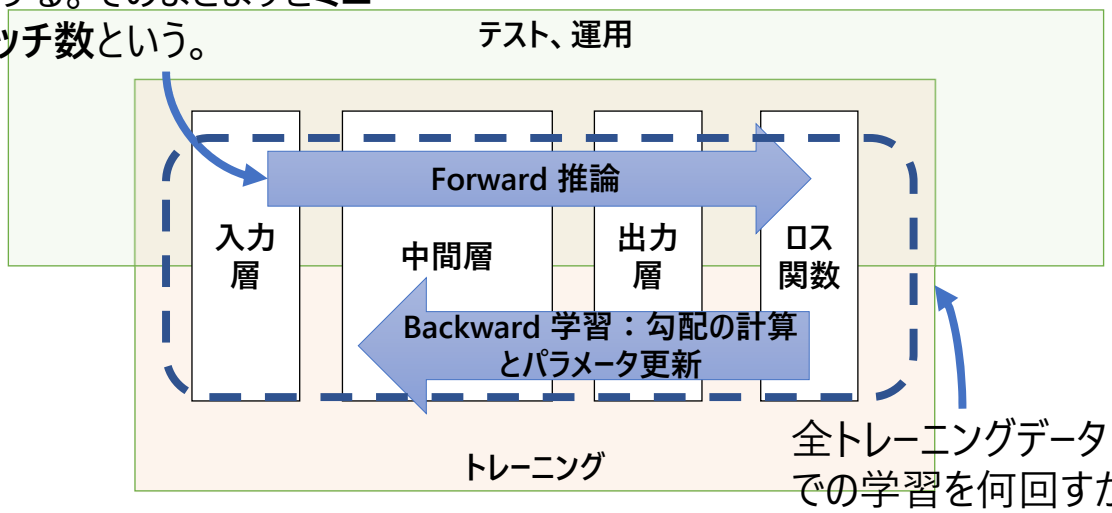
PyTorch事始め

課題：PyTorch 事始め

- 動的翻訳ができるChromeを使ってください。
- https://pytorch.org/ へ行ってください。
- 右クリックで、日本語へ翻訳。以下同様。
- 上のメニューからチュートリアル > ページ内の基本を学ぶ > 1. テンソルへ。
- Google Colabで実行を別タブでクリック。Colabページを翻訳すると、コードまで翻訳されてしまいますので、元のページの日本語訳説明を読みながら、Colabページを英語のままで、含まれるコードセクションを実行していきましょう。
- PyTorchのテンソルの動きを確認していきましょう。

エポック、ミニバッチ

データを何個かまとめて投入する。そのまとまりを**ミニバッチ数**という。



全トレーニングデータでの学習を何回すかを**エポック数**という

100本ノック第8章課題70～78と79

- 100本ノックの第8章は、中でも秀逸の課題群です。ニューラルネットの概念がステップバイステップで積み上げ式に学べます。

- ニューラルネットのコードに初めて触れる場合、最初から書こうとするのは無理です。まず、単純なサンプルを読み、慣れる必要があります。「100本ノック」の8章の課題 の70から78の回答例の、ポイントの部分だけ穴埋めにしています。「NLP、ニューラルネット.ipynb」というノートのサンプルを完成させてください。ノートには解説やコメントを入れています。

100本ノック第8章課題70～78と79

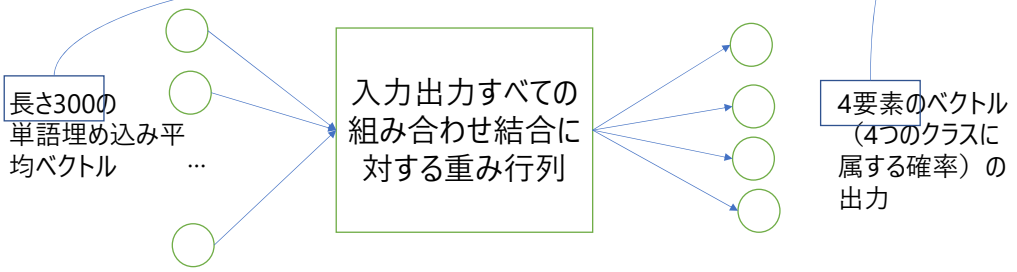
- 最後の79は、いろいろいじってみよという自由課題になっています。チャレンジしてみましょう。講師のコードも入れておきますが、いろいろいじってみてください。変更後のコードと、実行ログを残してください。この世界は、数理科学というより、まだ経験科学であるという実感を持てると思います。
- この章の到達点は、テキストを分類するタスクなので、応用が広いです。
- ここで、ニューラルネットの大枠は習得済み。ニューラルネットの入門書を一冊理解したことに相当します。

100本ノック第8章課題70～78と79

- あと、CNN、RNN、Attention、Transformerは、ネットアーキテクチャのデザインパターンを習得していくような感じです。第8章が終われば、もう自力でそれらのDeep Learningの技術を深めていくことができます。
- なお、自然言語アプリを目指す方は、第7章の「単語ベクトル」は必須です。

課題を解く参考

71：torch.nn.Linear(入力サイズ,出力サイズ)



実際には、データ件数（ミニバッチの数)分、まとめて計算され、
入力も出力も、1次元ベクトルではなく、ベクトルを横（行）、データを縦（列）
に並べた行列となる

呼び出し可能オブジェクト：*outputs = model(inputs)* は、forward呼び出しに化ける

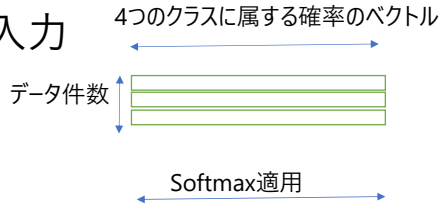
Pytorchは、Pythonの呼び出し可能オブジェクトの機能を利用している

- object.__call__(self[, args...])
  - Called when the instance is "called" as a function; if this method is defined, x(arg1, arg2, ...) roughly translates to type(x).__call__(x, arg1, ...).
  - オブジェクト()　と書くと、クラス.__call__(インスタンス、引数...)　に化ける

- Pytorchのコードを見ると
  - moduleクラスの__call__メソッドは、中で forward 呼び出しを行っている。

torch.softmax(x, dim=1)

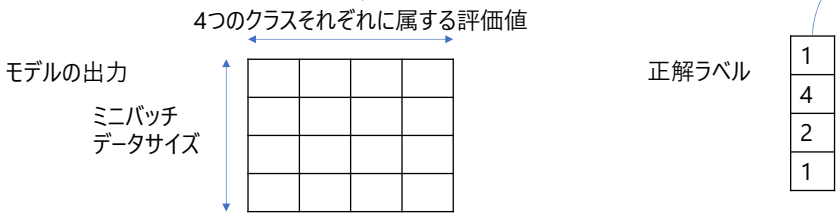
入力



softmaxをとる次元：dim=0がミニバッチ、dim= 1 が4要素のベクトル。dim=1とすることで、4個の値に対して合計1になるように変換する

72. torch.nn.CrossEntropyLoss()

loss = CrossEntropyLossのインスタンス(モデルの出力, 正解ラベルの1次元テンソル)



確認クイズ

- 確認クイズをやってください。