自然言語処理
—CNN —

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

100本ノック第9章とCNNの位置づけ

- １００本ノック課題集第9章は、RNN、CNN、Transformerを扱っている。
- CNNは、現在、画像処理系では基本となる技術である。100本ノックの課題では、自然言語処理へ応用する事例が載せられている。

RNNは、時系列データの処理技術として、音声認識や機械翻訳のDeep Learningの基幹技術だったことがあります。
しかし、いまのところ、Transformerにその位置を奪われたかのような感じとなっています。
ただ、RNNは、Transformerでも重要なSeq2SeqやAttentionなど関連技術の生まれる母胎だったので、概略だけ見ていきます。

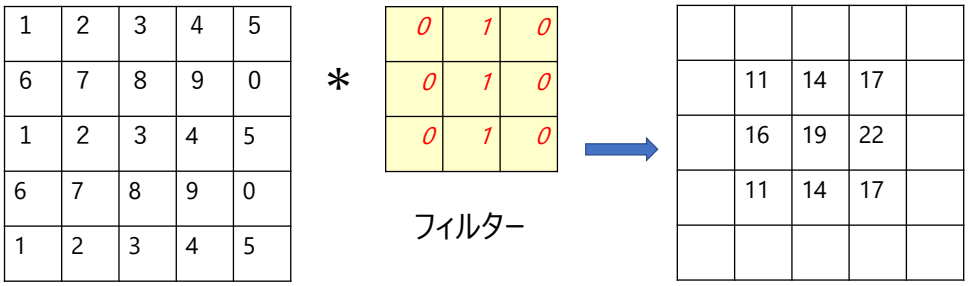
自然言語処理
CNN

解説動画

ここではCNNについてみていきます。

Convolution層　(CNN)

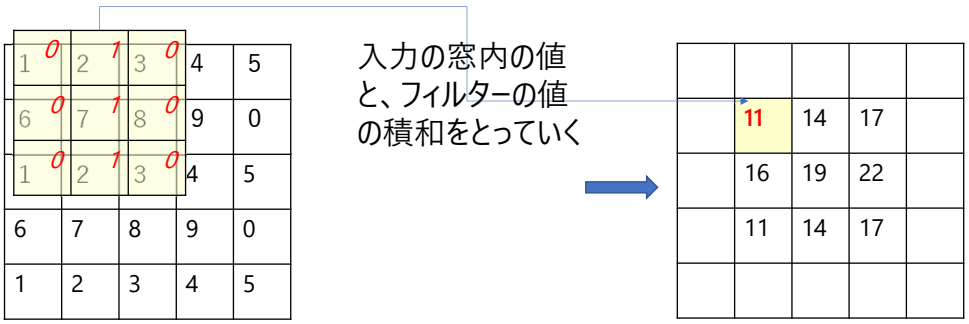
特徴を抽出する(2次元データの場合)



入力　　出力

CNNは、Convolutional Neural Networkの略称です。Convolutionは、画像処理で畳み込み操作を意味します。
入力が2次元の画像だとします。これに対し2次元の小さなフィルターを準備し、入力から何等かなお特徴を抽出していきます。

Convolution層



入力に対し、フィルターを重ねて、フィルターの窓内で、入力とフィルターのセルの値の積和をとっていきます。

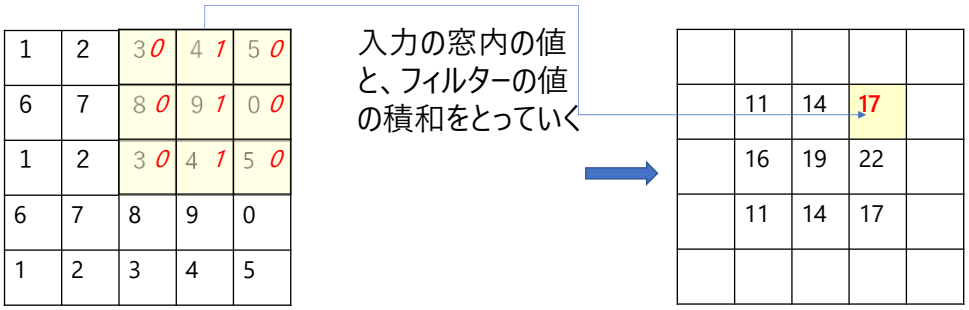
Convolution層

1	2 *0*	3 *1*	4 *0*	5
6	7 *0*	8 *1*	9 *0*	0
1	2 *0*	3 *1*	4 *0*	5
6	7	8	9	0
1	2	3	4	5

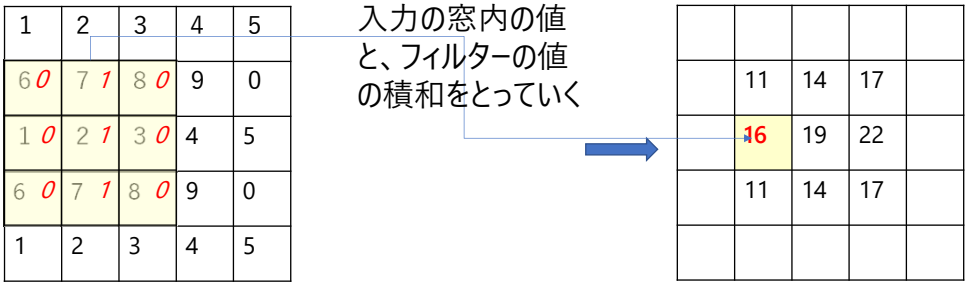
入力の窓内の値
と、フィルターの値
の積和をとっていく

	11	14	17	
	16	19	22	
	11	14	17	

Convolution層

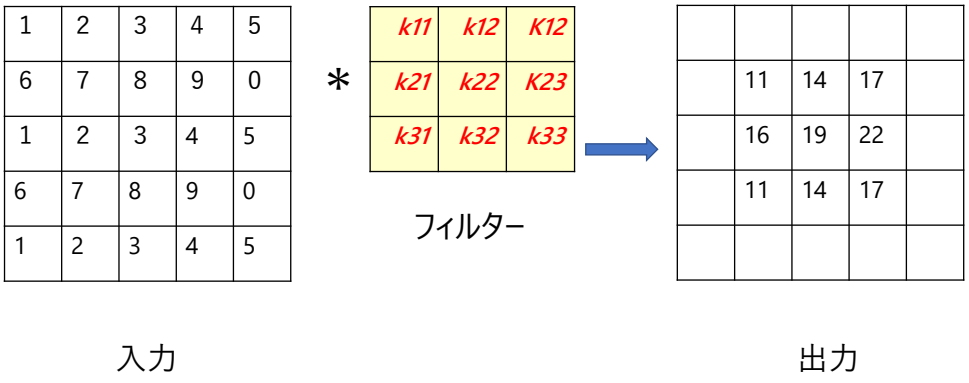


Convolution層



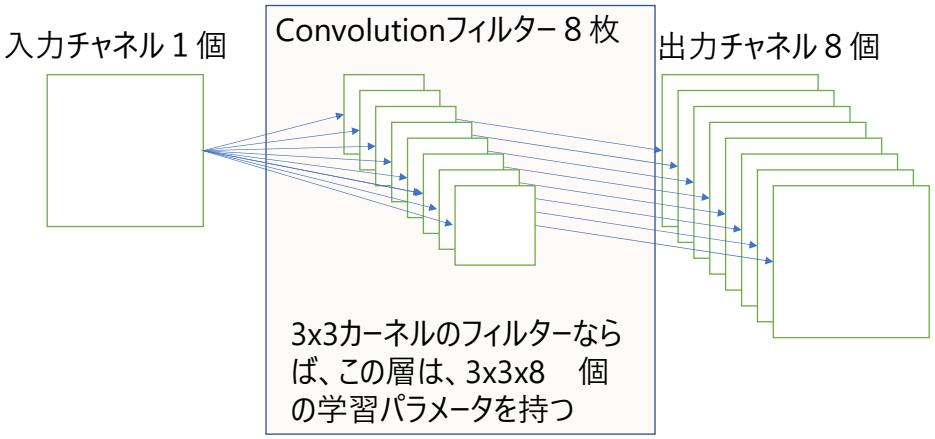
以下省略...

フィルタの各要素が学習パラメータ



このようなフィルターの値は、学習パラメータとして、逆伝播学習におよって、ネットワーク全体で望ましい最終結果が得られるように最適化されます。ちょうど全結合の重みに対応します。

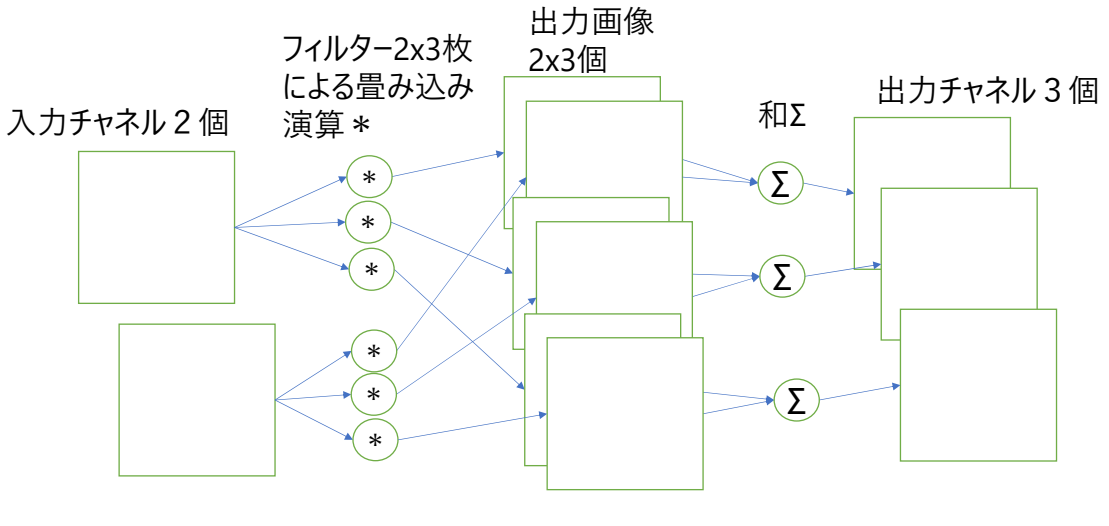
Convolution層の入出力チャネルとパラメータ



フィルターは何らかの特徴を抽出します。
フィルターは通常何枚か利用されて、特徴抽出を分担します。
あるレイヤで、入力の画像枚数を入力のチャネル数、出力の画像数を出力チャネル数といいます。
元の入力画像がGrayScaleだとすると、最初のCNNレイヤの入力チャネル数は1です。
元の入力画像がRGBだとすると、最初のCNNレイヤの入力チャネル数は３です。
CNNレイヤを複数重ねる場合、通常、前段の出力チャネル数が後段の入力チャネル数になります。

ここで、あるレイヤが、フィルターの枚数が８で、３ｘ３のフィルターだとした場合、そのレイヤは3x3x8の学習パラメータを持つことになり、
それらが逆伝播学習で更新されていきます。

入力チャネルが複数の場合



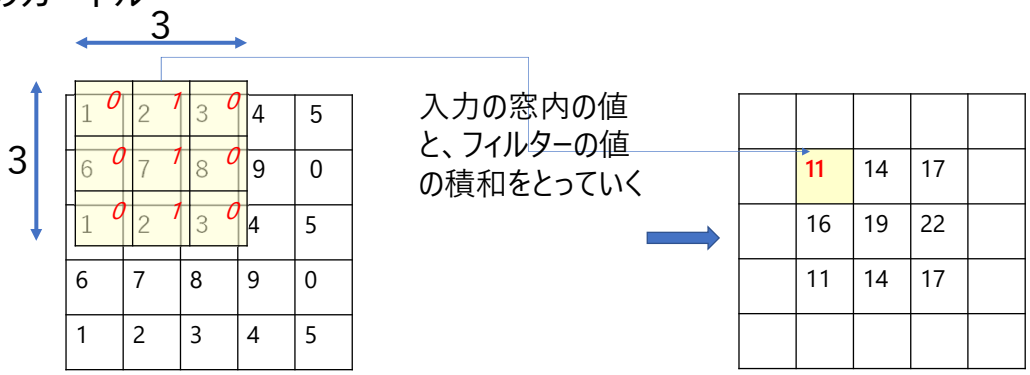
CNNはこのように単純な演算ですが、入力チャネルが複数の時の動きは
ちょっとだけ複雑になります。
入力チャネルが２個で、出力チャネルが3個だとします。
まず、２ｘ３の組み合わせ数のフィルターが6枚の画像を出力します。
次に、出力チャネルに対応する画像を2枚づつ加算して、3枚の画像を出力します。

これは、全結合の考え方とそっくりです。
全結合では入力２ノードと出力３ノードの場合、２ｘ３の組み合わせの結合線に重みがあって、
入力にそれら重みをかけたものの和をとることで３つの出力を計算しました。

実は大規模なネットワークでは、１ｘ１のフィルターというものが用いられることがあるのですが、
その場合は、入力画像と出力画像の全結合そのものとなります。１ｘ１フィルターは、チャネル数を調整するために
用いられたりします。
それを理解するためにも、ＣＮＮというのは全結合を拡張したみたいなものだと理解しておくと、わかりやすいです。

カーネルサイズ

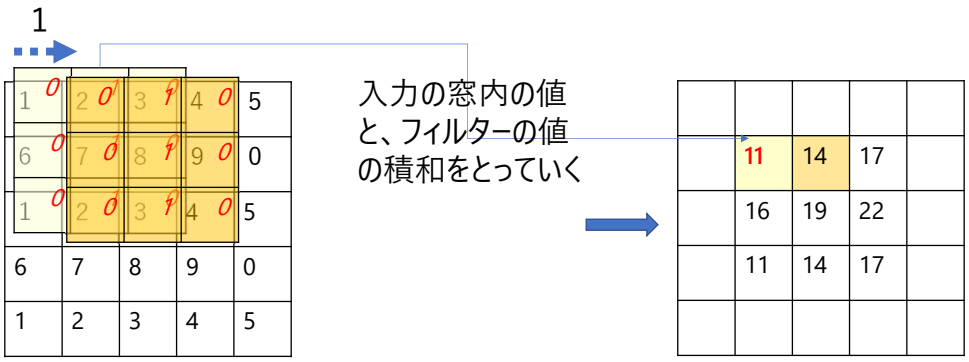
3x3のカーネル



カーネルサイズが大きいほど、グローバルに観察できるが、学習パラメータが増える

いくつか用語を見ていきます。
フィルターの大きさをカーネルサイズといいます。
カーネルサイズが大きいほど、グローバルに観察できるが、学習パラメータが増え、計算量も増えます。

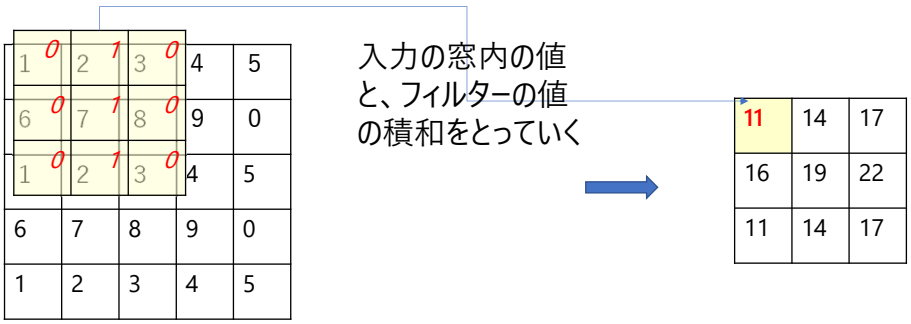
ストライド（歩幅、移動幅）



ストライドが小さいほどきめ細かに観察するが、計算量が増える。

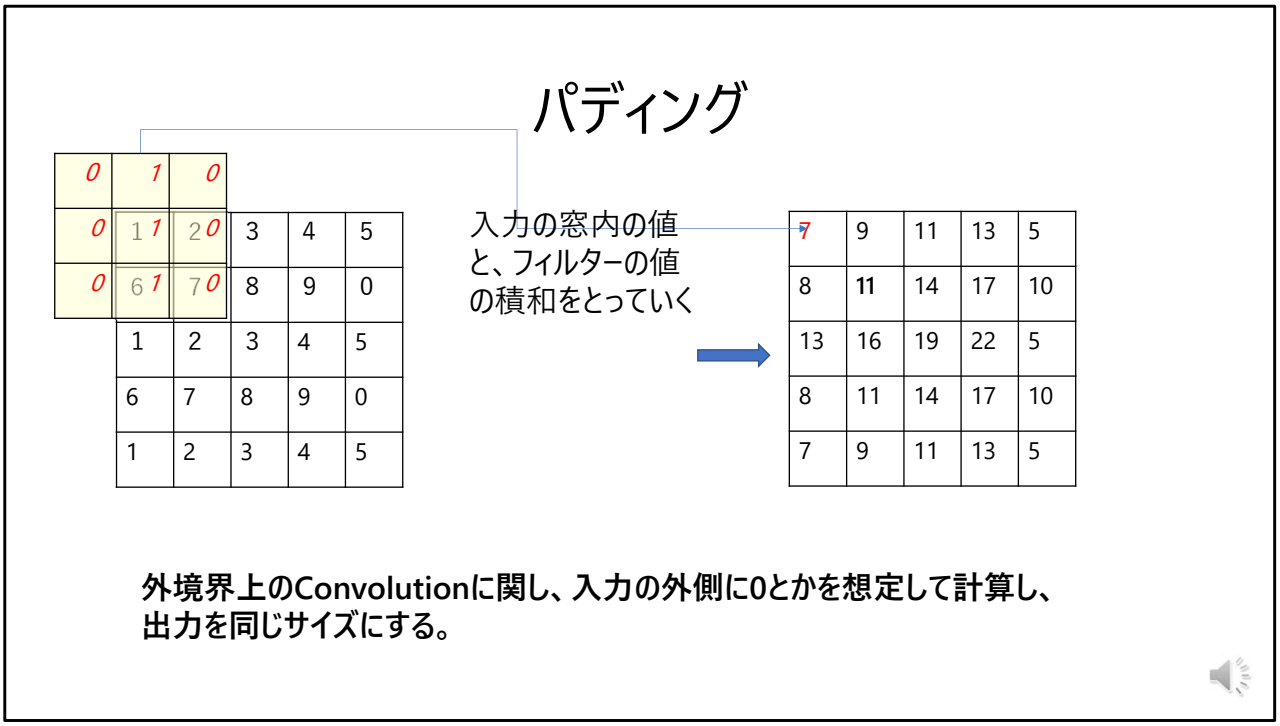
ストライドとは、フィルターをかけていくときの移動幅のことです。
ストライドが小さいほどきめ細かに観察することになりますが、計算量が増えます。

パディング

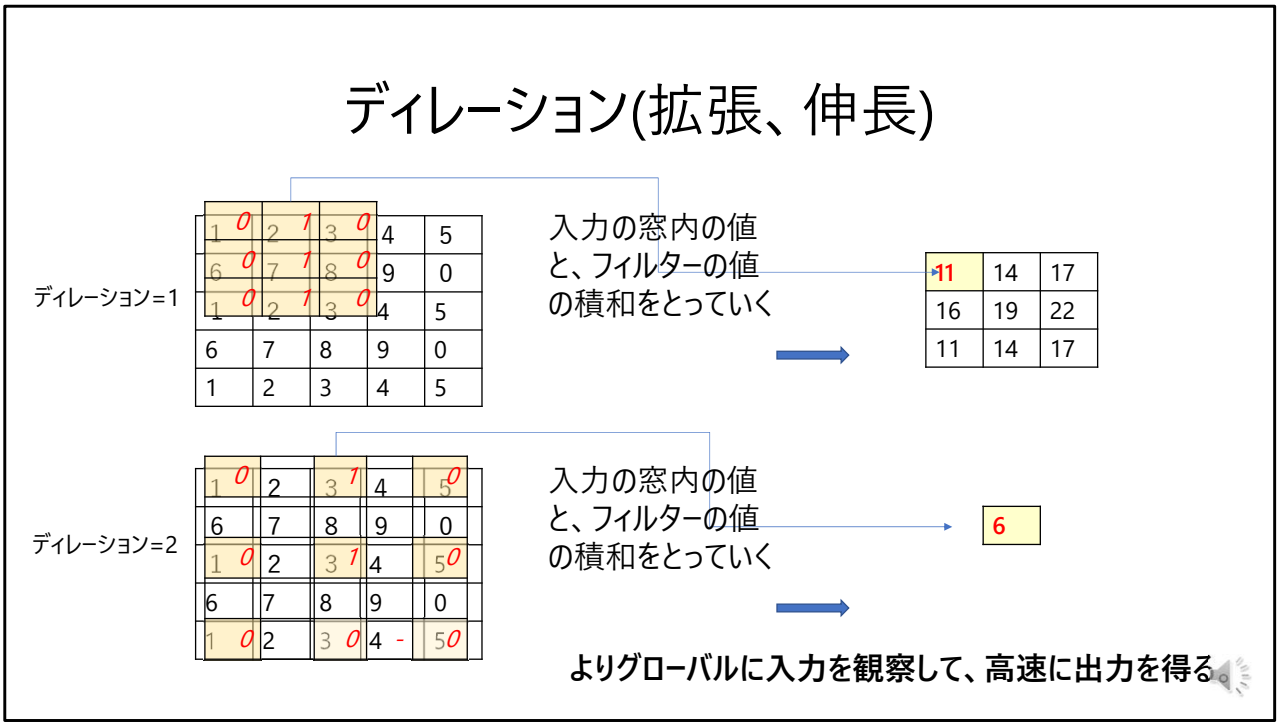


Convolutionはデフォルトで出力は小さくなる。

CNNというのは窓内のセルの中心の値を求めていくので、入力サイズに比べて出力画像は小さくなります。
５ｘ５の入力に対して、３ｘ３のフィルターを用いれば、出力は５－２の、３ｘ３となります。



そこで、入力画像の外側に、０とかのセルを仮定し、入力画像セルのすべてにあまねく出力地を得る手法がパディングです。
出力画像のサイズを、入力画像のサイズと同じにしたい場合、このPaddingを用います。



Dilationとは、フィルターの適用するセルを1個置きや2個おきにするという考え方です。
Dilation＝１の場合は、隙間のないフィルターとなります。
Dilation＝２の場合は、2つ目のセルを見る、1個スキップしてみるフィルターとなります。
Dilation＝３の場合は、３つ目のセルを見る、２個スキップしてみるフィルターとなります。
よりグローバルに、高速に出力を得ることができます。

CNNの貢献と今

- CNNはDeep Learningの発展を支えた。
- 今、Self-AttentionがCNNに影響を及ぼしている。

CNNが出てくる前、画像処理におけるフィルターは、人が設計していました。

しかし、フィルターを機械に学習させるアプローチが出て以来、CNNはDeep Learningの発展を支えました。

例えば、それまでは込み入った数式と特殊なノウハウの塊だった手書き文字認識が、以来、Deep Learningの入門教科書通りに作れば、だれでも書けるものになった。

今、Self-AttentionがCNNに影響を及ぼしつつあい、CNNも変貌しつつあります。

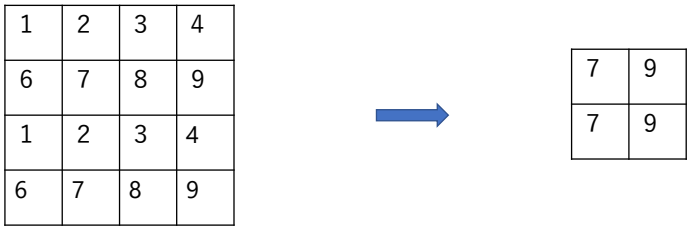
自然言語処理

MaxPooling

解説動画

ここでは、MaxPoolingについてみていきます。

MaxPooling

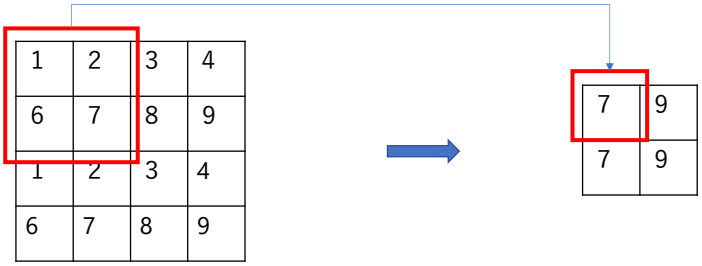


入力　　出力

MaxPoolingというのは、近傍の特徴を集約するテクニックです。

MaxPooling

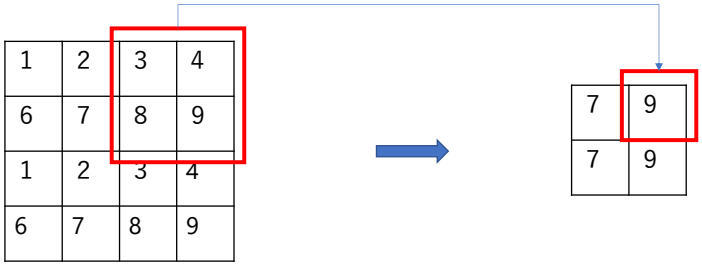
最も大きな値を拾っていく



ある窓サイズで、入力画像のピクセルをスキャンしていきます。
窓の中の最大値を拾い、出力します。

MaxPooling

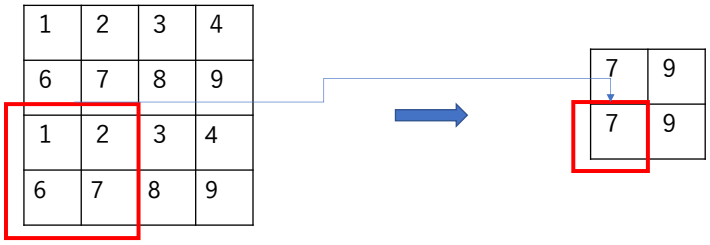
最も大きな値を拾っていく



これを、窓を移動させながら繰り返します。

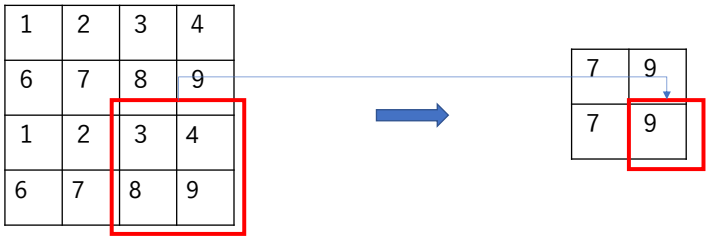
MaxPooling

最も大きな値を拾っていく



MaxPooling

最も大きな値を拾っていく



MaxPooling

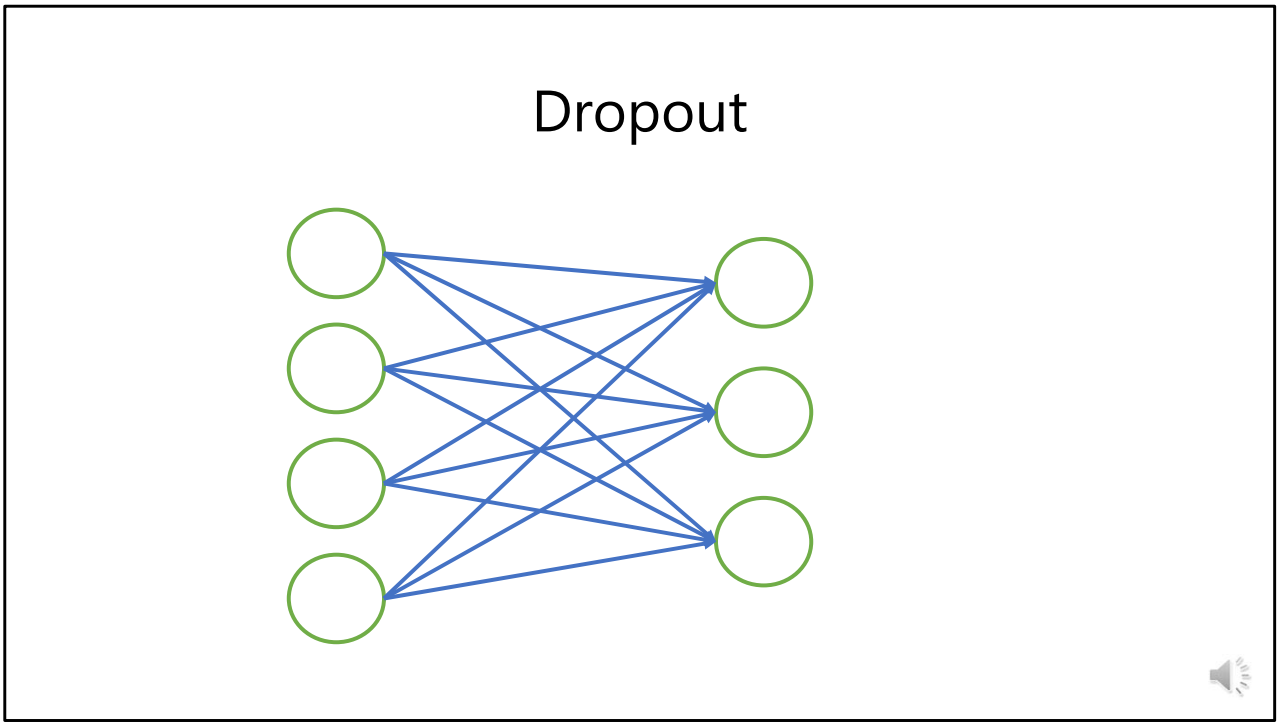
- 広い範囲の情報を集約する。
- データを小さくして後続の計算量を小さくする。

このテクニックは、近傍の情報を集約することで、より広い範囲の情報の集約につなげます。
最大との代わりに、平均をとる方法もあります。
出力画像が小さくなるので、以降の計算量を抑えることができます。

自然言語処理
Dropout

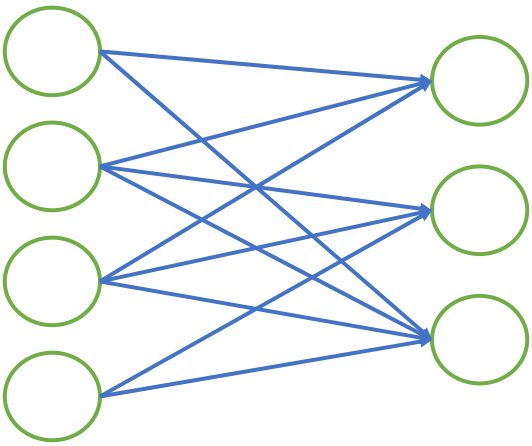
解説動画

ここではDropoutというテクニックについてみていきます。

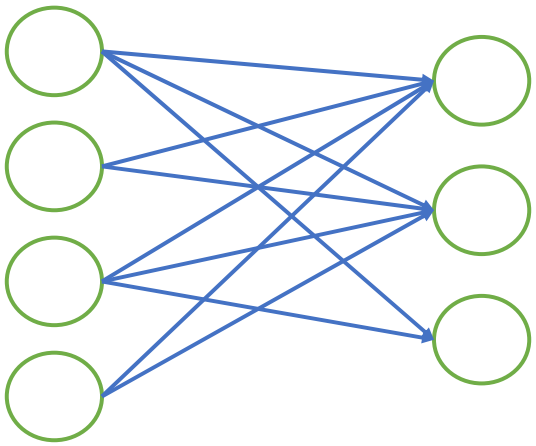


Dropoutとは、入力から出力を計算する際の経路が多数ある場合、そのうちの何割かをランダムに切ります。

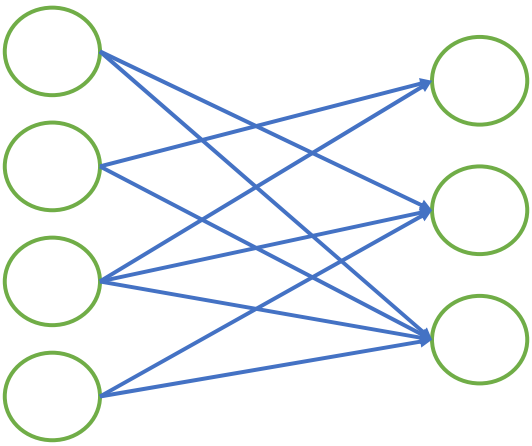
Dropout



Dropout



Dropout



Dropout

仮想的にサブネットワークを構成しそれらの多数決で出力を得る、感じにすることで、般化能力を高める、と言われる（経験的）。

Dropoutテクニックは、いわば、仮想的にサブネットワークを構成しそれらの多数決で出力を得る、感じにすると解釈されています。
学習能力を高めたり、過学習を小さくするなどの効果があると言われます。
理論的な解析よりも、経験的に有効だとされています。

自然言語処理

Batch Normalization

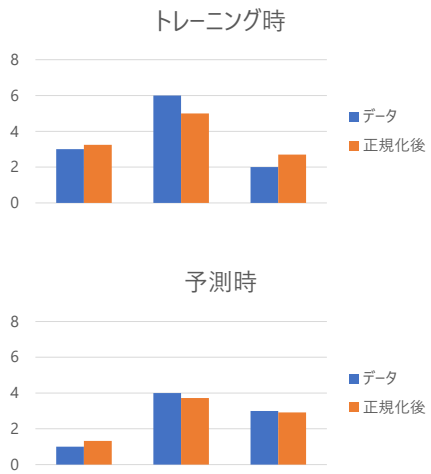
解説動画

ここでは、バッチNormalizationというテクニックを見ていきます。

Normalizationとは？

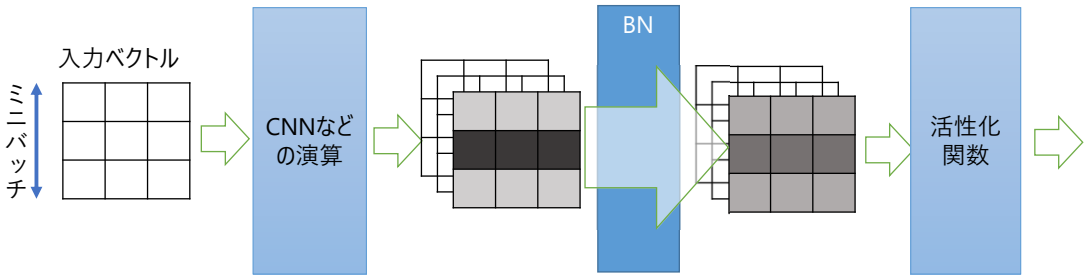
- 入力データや推論中間データが、トレーニング時と予測時とで、分布が大きく異なると困る。
- ある範囲のデータに関して、分布の偏り（凸凹）を小さくする。

$$\mu = \frac{1}{m}\sum_i H_{i,:},$$
$$\sigma = \sqrt{\delta + \frac{1}{m}\sum_i (H-\mu)_i^2}$$
$$H' = \frac{H-\mu}{\sigma},$$



データの正規化が、なぜ必要か。
データが、トレーニング時と予測時とで、分布が大きく異なると困ります。
そういう場合、トレーニングデータで過学習すると、予測時に使い物になりません。
そこで、データに正規化をかけて、ばらつきを抑えることがあります。

Batch Normalization



バッチNormalizationというのは、同時並行して処理するデータであるミニバッチごとに、
ミニバッチ内のデータの平均と分散を求めて、ばらつきを少し抑えたデータに変換します。
これを、あるレイヤの出力と活性化関数に通す中間に挟みます。

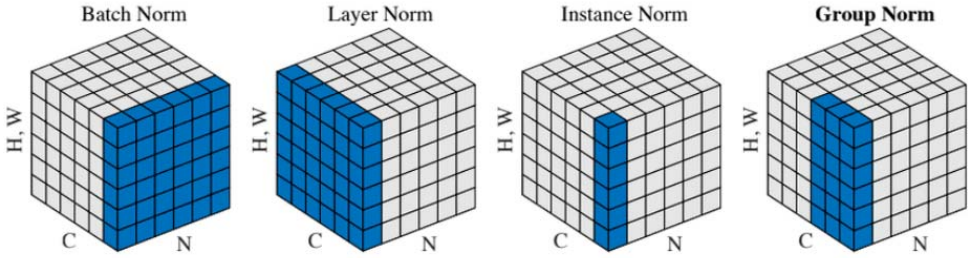
Batch Normalization

- データのばらつきを抑えることで、学習が高速化し、過学習にも効果がある、と言われる。

バッチNormalizationは、非常に強力であるといわれます。
学習が高速化し、過学習を抑える効果があるといわれます。
Dropoutの代わりに、Batch Normalizationを使ったりします。

データの平均と分散をとって補正する

以下、入力データをシリアライズし（H,W）、チャネル（C）、ミニバッチ（N）との3次元立体で示す



ミニバッチ内の正規化

チャネル内の正規化

１入力データ内の正規化

特定のチャネルグループ内の正規化

https://arxiv.org/pdf/1903.10520.pdf

Normalizationには、いくつものVariationがあります。
ミニバッチ単位で正規化するものを、バッチNormalizationと呼びます。
チャネル内のデータに関して正規化するものを、レイヤNormalizationと呼びます。Transformerが利用しています。
そのほか、インスタンスNormalization、Group Normalizationなどがあります。

MNISTサンプルコード

MNISTサンプル

- 自然言語処理ではなく、画像処理になりますが、ディープラーニングの典型的な流れと構成要素の概略を把握するために、いいサンプルですので、見ておきます（ニューラルネットの入門書で最終章に用いられていたりする）。
- これは、CNNを利用しています。自然言語処理でもCNNを使うことがあります（例：100本ノック課題８６）。

課題MNISTサンプルの理解

- Mnist_sample.ipynbをダウンロードして、Google Colaboratoryで動かしてみてください。
- 最初のコードは、PyTorchのサンプル集の中の https://github.com/pytorch/examples/tree/master/mnist （もとのPyTorchのコードは、煩雑になりがちなデータ回りの処理もクラス化されているため、Mnistの主要な処理構造が見えやすいものとなっています。）を、教育用に単純化したものです。後続のコード解説、コードのコメント、PyTorchのマニュアルを参照しながら、このコードを咀嚼して下さい。
- そのあとには、参考のため、Convolution結果を表示したり、モデルを変えたコードを入れてあります。

MNistサンプルコードを読むための追加情報

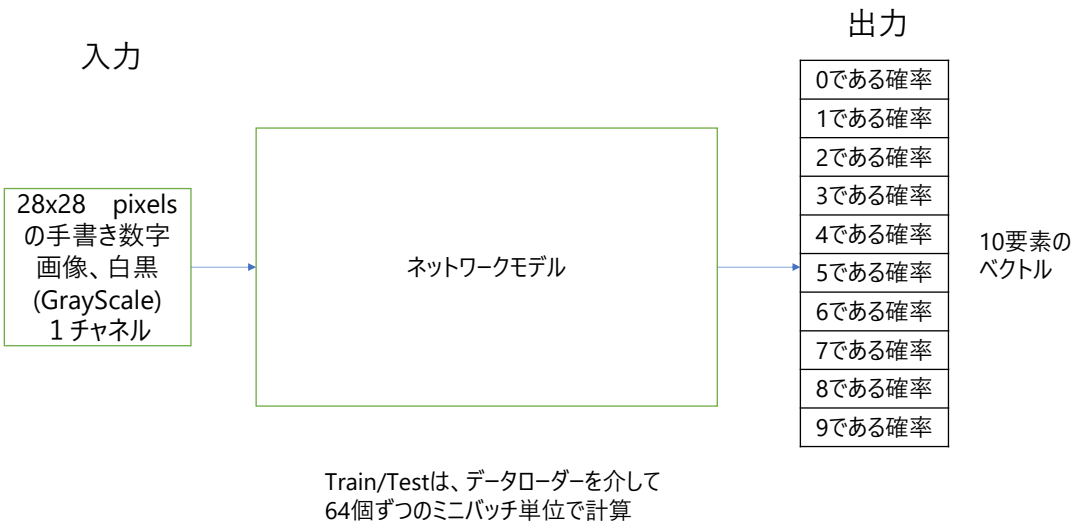
PyTorchマニュアルの読み方

- https://pytorch.org/docs/stable/index.html　へ行ってください。
- 左上に検索ボックスがあるので、そこにわからない書き方をタイプしてください。
- 各ページは英語です。Chromeの場合、該当ページ上で、右クリックで、「日本語に翻訳」を選んでください。

torch.nnとtorch.nn.functionalの違い

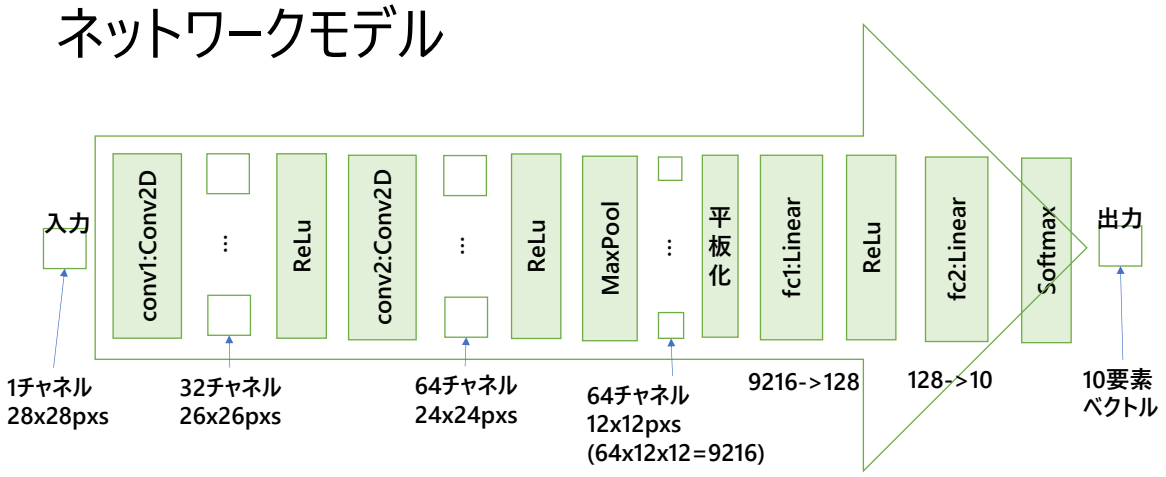
- nn　モジュール
  - クラス定義
  - 例えば、ちょっと複雑な初期設定したい場合、オブジェクトを作っておく。
    - self.conv1=nn.Conv2d(1,32,3,1)
      - nn パッケージ内の　Conv2dクラスのオブジェクトを作成し、初期設定する。
    - x=self.conv1(x)
      - conv1オブジェクトを入力データに対し作用させる。Pythonの呼び出し可能オブジェクトの機能を使い、__call__している。
- functional　モジュール
  - nn　クラスの機能を、関数として利用できるようにしたもの。
  - 例えば
    - x=F.relu(x)
    - x=F.max_pool2d(x, 2)
      - nn　モジュールの MaxPool2dのオブジェクトを作成し、引数で初期化して、入力データに作用させる。

Mnist サンプルの入出力

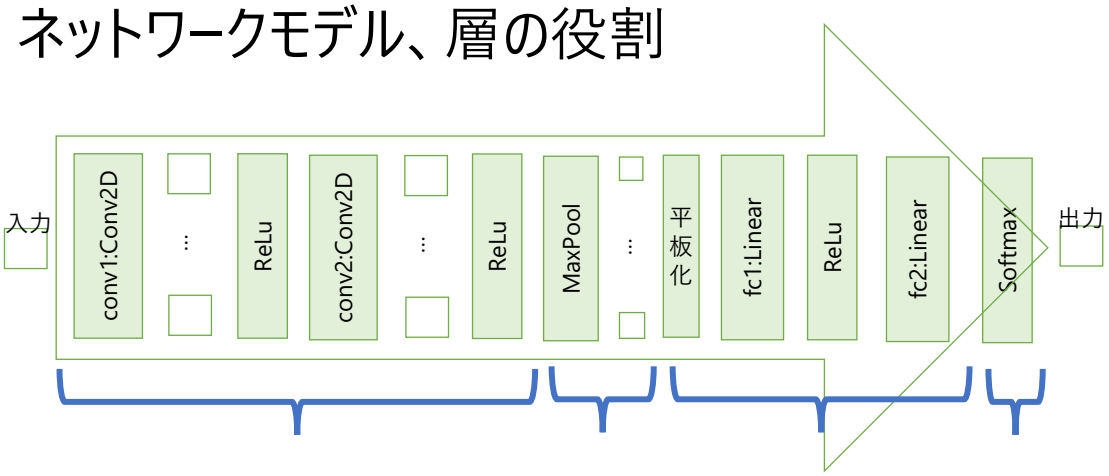


Train/Testは、データローダーを介して
64個ずつのミニバッチ単位で計算

ネットワークモデル


入力
conv1:Conv2D
ReLu
conv2:Conv2D
ReLu
MaxPool
平板化
fc1:Linear
ReLu
fc2:Linear
Softmax
出力
1チャネル
28x28pxs
32チャネル
26x26pxs
64チャネル
24x24pxs
64チャネル
12x12pxs
(64x12x12=9216)
9216->128
128->10
10要素
ベクトル

ネットワークモデル、層の役割



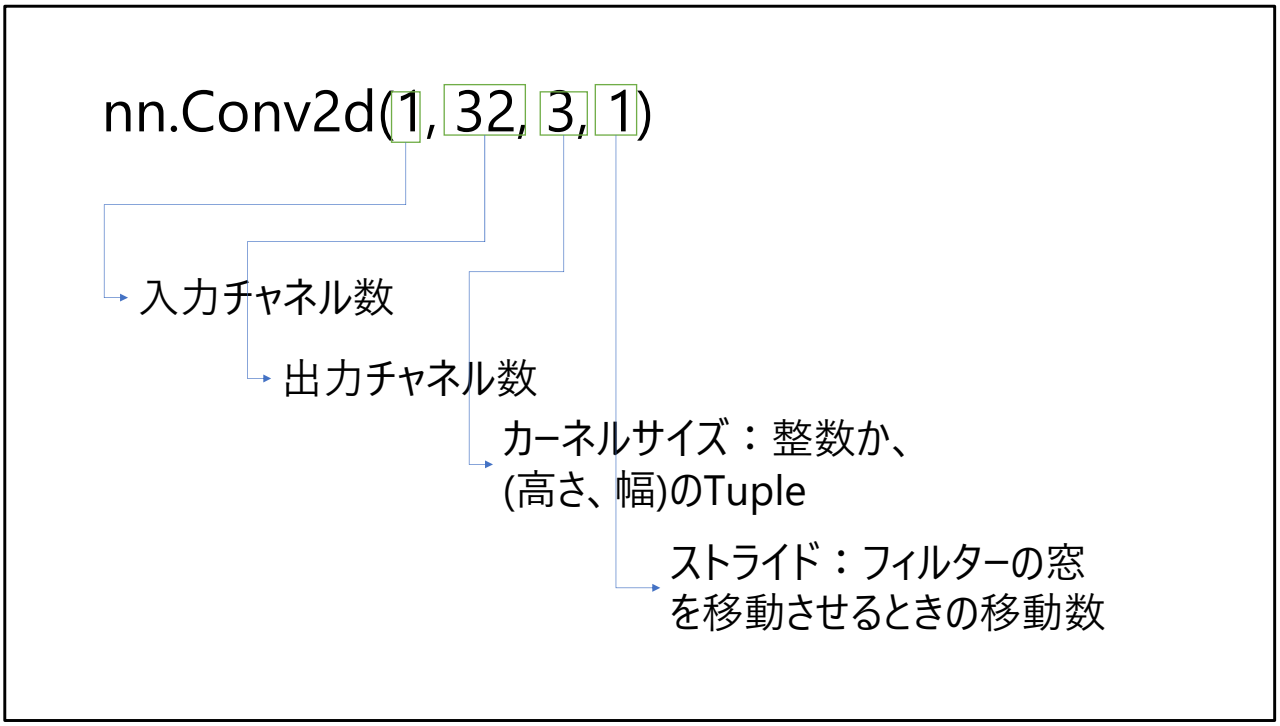
CNN2層で画像の何らかの特徴を抽出

サイズを小さくし、グローバルに特徴をまとめる

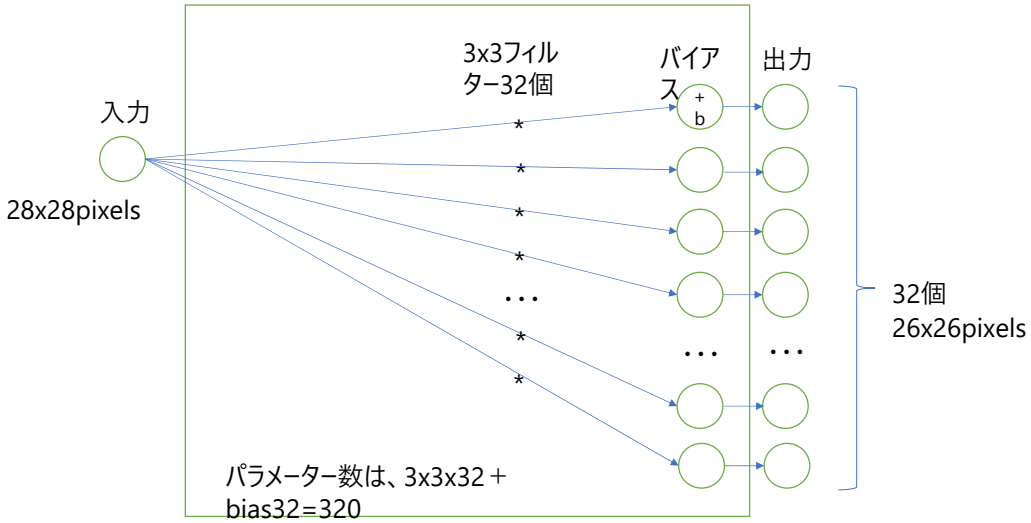
全結合2層で情報の取捨選択・集約

確率に変換

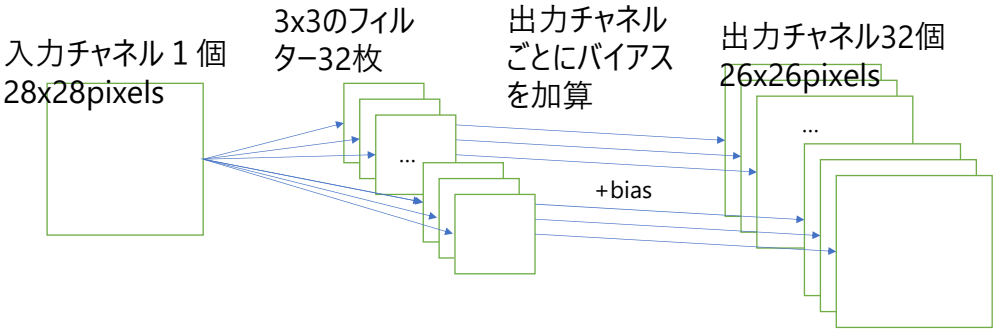
Netクラス__Init__


nn.Conv2d(1, 32, 3, 1)
入力チャネル数
出力チャネル数
カーネルサイズ：整数か、
(高さ、幅)のTuple
ストライド：フィルターの窓
を移動させるときの移動数

conv1

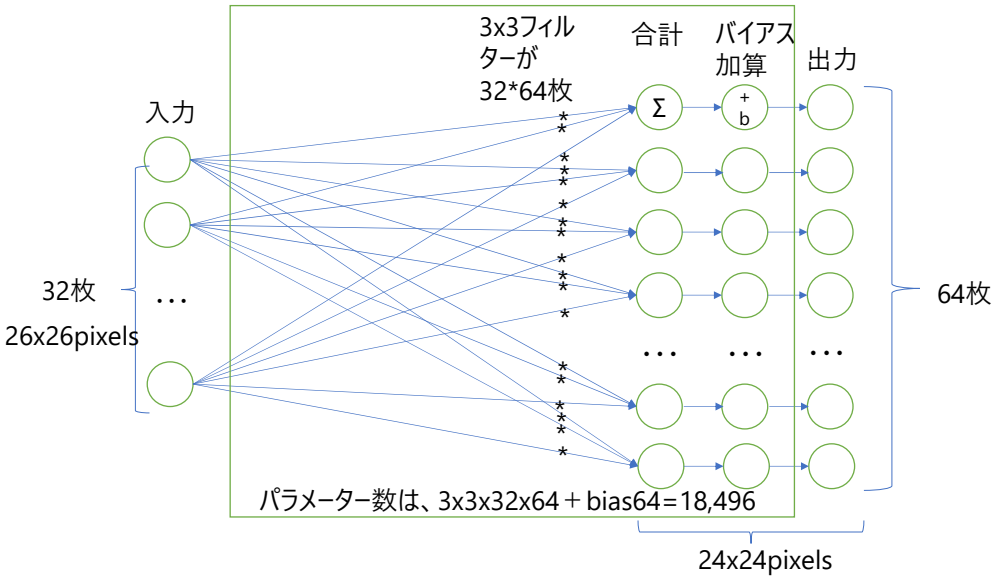

入力
28x28pixels
3x3フィルター32個
バイアス
+
b
出力
32個
26x26pixels
パラメーター数は、3x3x32＋bias32=320

conv1



パラメーター数は、3x3x32＋bias32=320

conv2


3x3フィルターが32*64枚
合計
バイアス加算
出力
入力
Σ
+b
32枚
26x26pixels
64枚
パラメーター数は、3x3x32x64＋bias64＝18,496
24x24pixels

nn.Conv2d(32, 64, 3, 1)

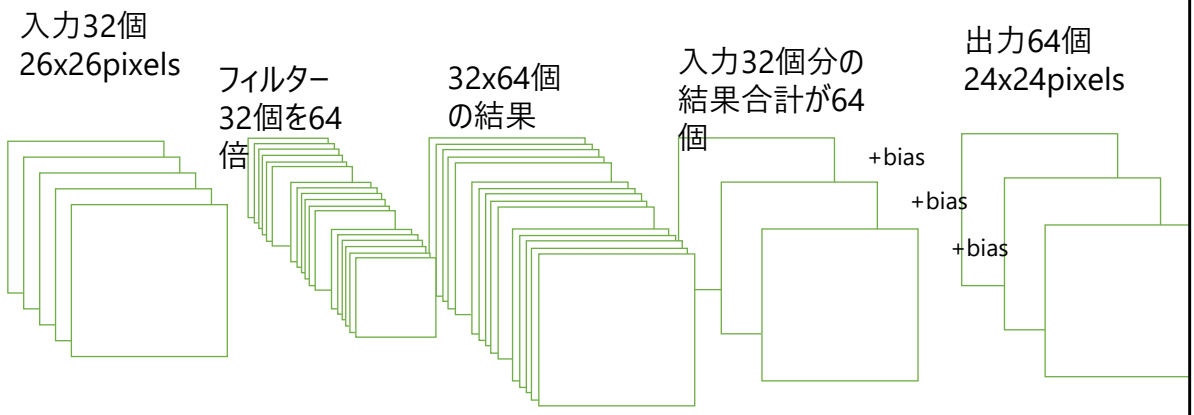
入力チャネル数

出力チャネル数

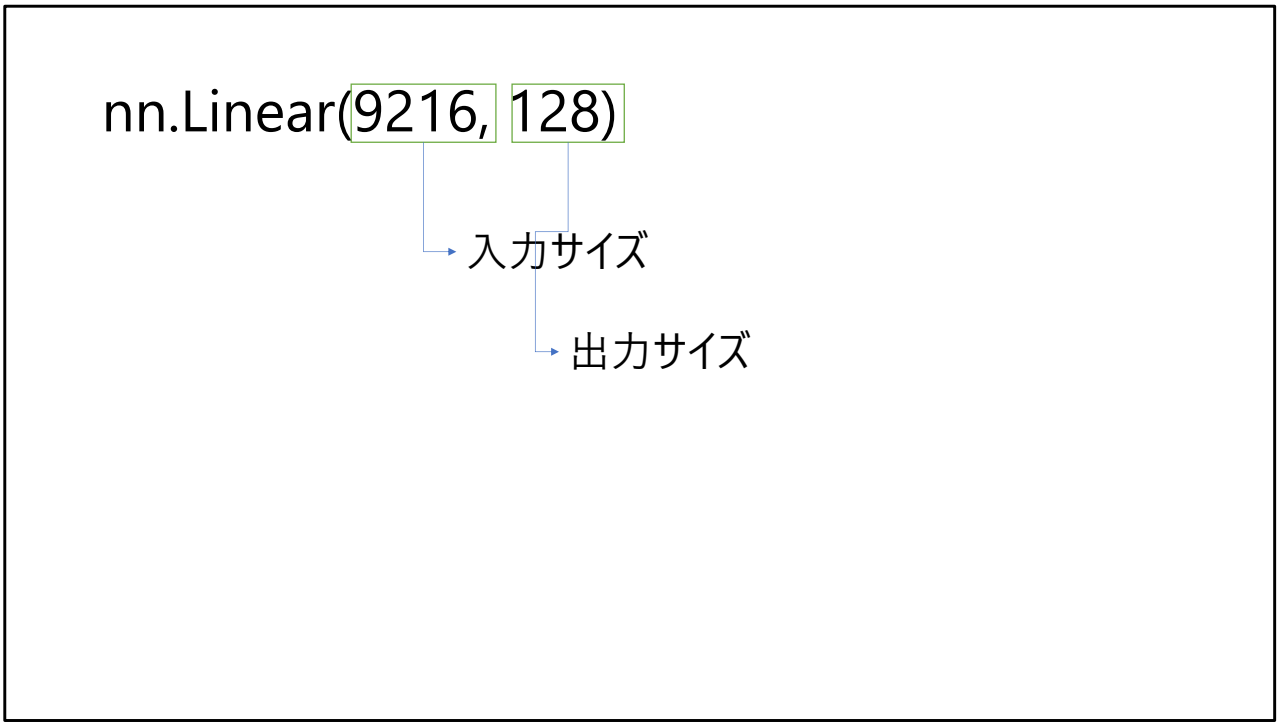
カーネルサイズ：整数か、
(高さ、幅)のTuple

ストライド：フィルターの窓
を移動させるときの移動数

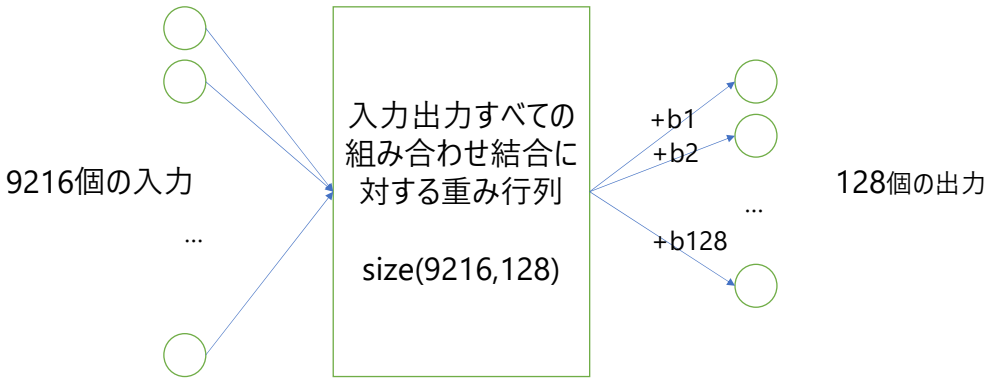
conv2



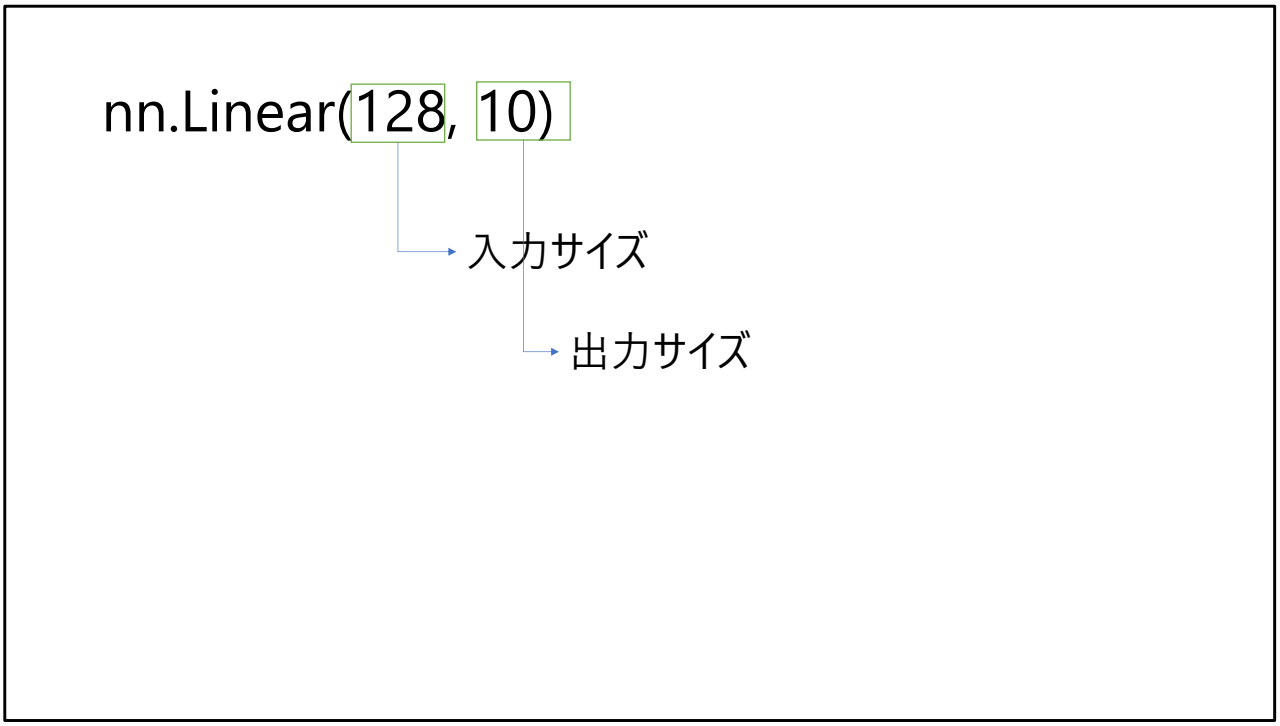
パラメーター数は、3x3x32x64＋bias64＝18,496


nn.Linear(9216, 128)
入力サイズ
出力サイズ

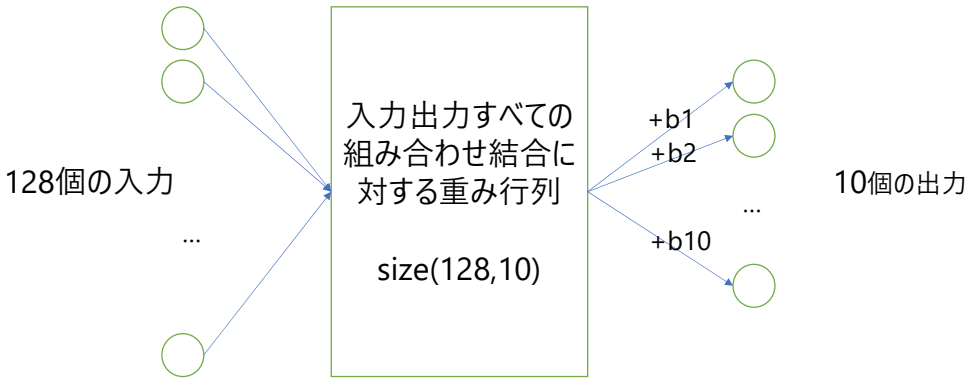
fc1



実際には、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、ミニバッチを行方向に収めた行列となる


nn.Linear(128, 10)
入力サイズ
出力サイズ

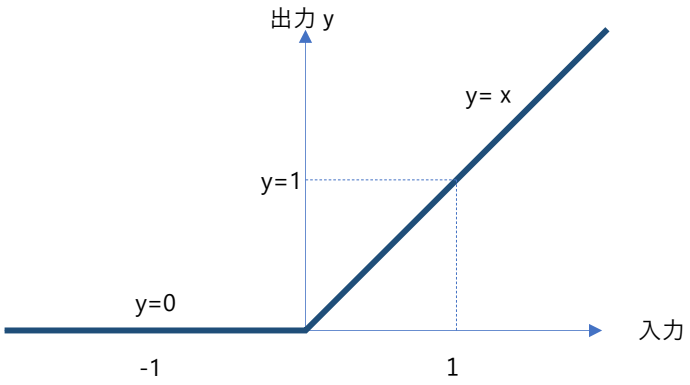
fc2



実際には、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、ミニバッチを行方向に収めた行列となる

Netクラス　forward

活性化関数ReLU(Rectified Linear Unit)



max_pool2D(x, 2)

入力： (ミニバッチ数、チャネル数、高さ、幅）の
sizeを持つテンソル

カーネルサイズ：整数
省略可能引数にストライド（ウィンドウの移動幅）があり、
ストライドのデフォルトはカーネルサイズ

MaxPooling：広い範囲の情報を集約する。
データを小さくして計算量を小さくする。

1	2	3	4
6	7	8	9
1	2	3	4
6	7	8	9

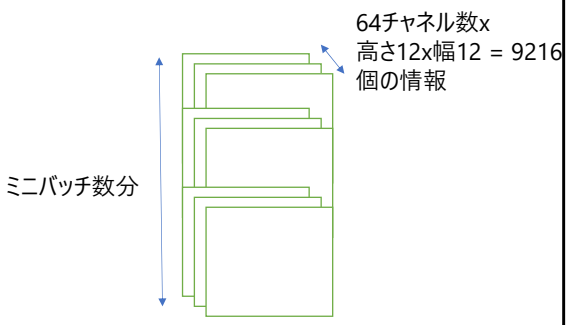
入力

7	9
7	9

出力

torch.flatten(x, 1)

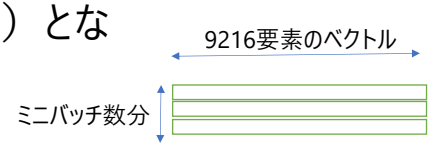
入力テンソル：ここでは、sizeは (ミニバッチ数、チャネル数、高さ、幅）



平板化する最初の次元１　-> size が (ミニバッチ数、平板化されたデータ）となる。

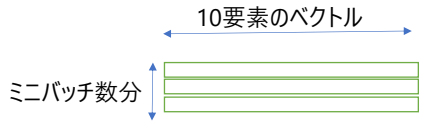
tensor([[...],[...],....,[...]]

ミニバッチの数だけ

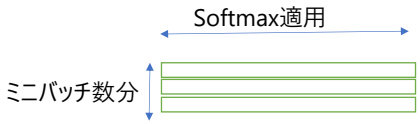


F.softmax(x, dim=1)

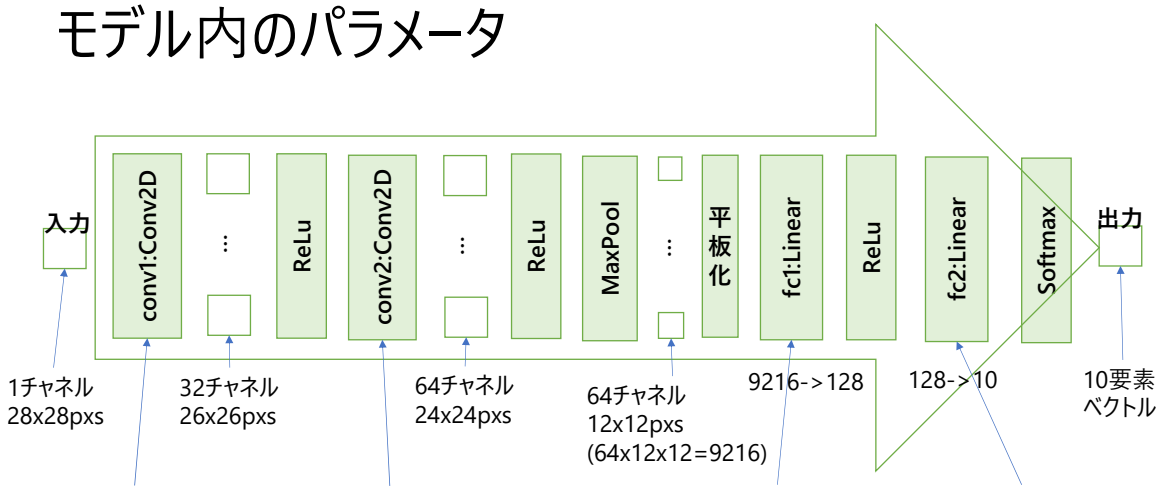
入力



softmaxをとる次元：0がミニバッチ、1が10要素のベクトル



モデル内のパラメータ

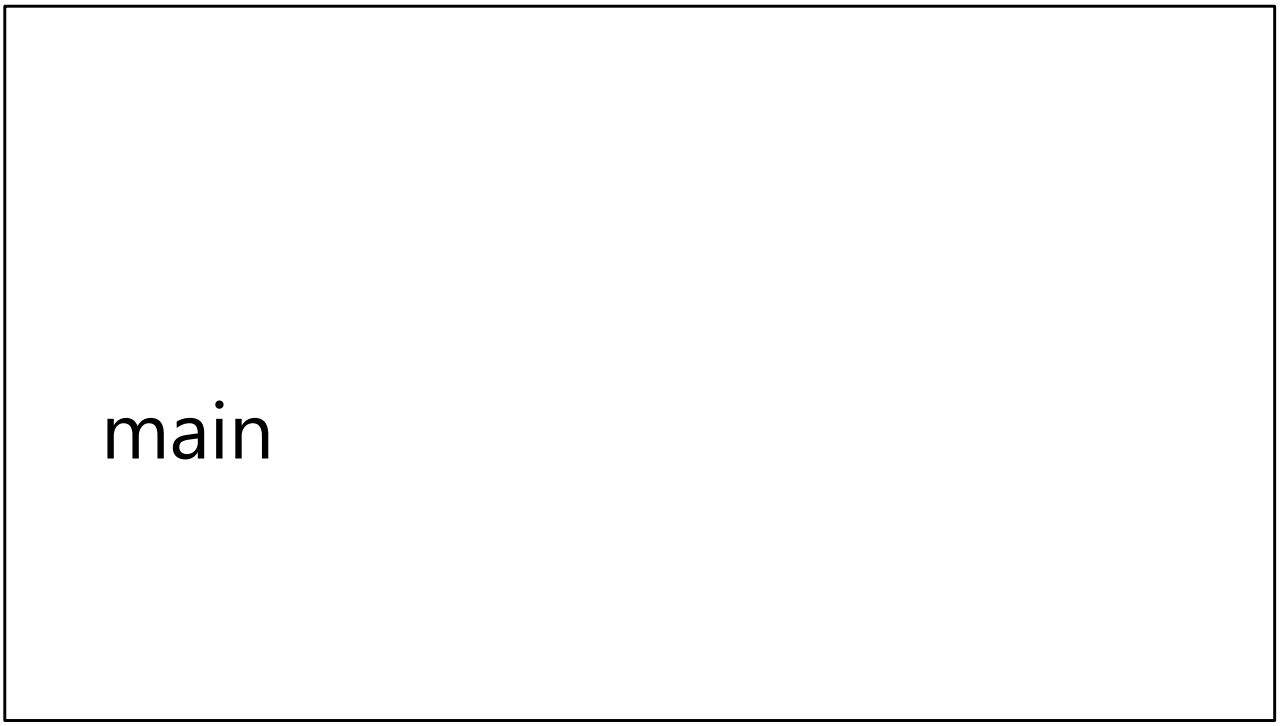


パラメーター数
3x3x32＋bias32
=320

パラメーター数
3x3x32x64＋bias64
=18,496

パラメーター数
9216x128+bias128
=1,179,776

パラメーター数
128x10+bias10
=1,290


main

データセット、データローダー

- データセット
  - 入力データを、Pytorch Tensor形式に変換したり、値を正規化したり、など変換してから、送る。
- データローダー
  - ミニバッチ単位にまとめて、モデルに送る。

```
optim.SGD(model.parameters(), lr=0.1)
```

モデルのパラメータ

学習レート

SGD (Stochastic Gradient Decent) 確率的勾配降下法。ほかにも。

勾配降下法なのだが、入力データはシャッフルしてあってランダムなので、確率的。

SGD以外にいろんなOptimizerが準備されている。

StepLR(optimizer, step_size=1, gamma=0.7)

エポック何個で学習具合を低減させるか

学習レートの逓減率

エポックが変わったときに、段階的に、一律に学習レートを低減させる。
ほかに低減させるやり方のOptionが何個か準備されている。

train, test

ミニバッチ

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

データを何個分かまとめる

ミニバッチ
３個

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

GPUの能力活用

outputs = model(inputs) が、forward呼び出しに化ける訳

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

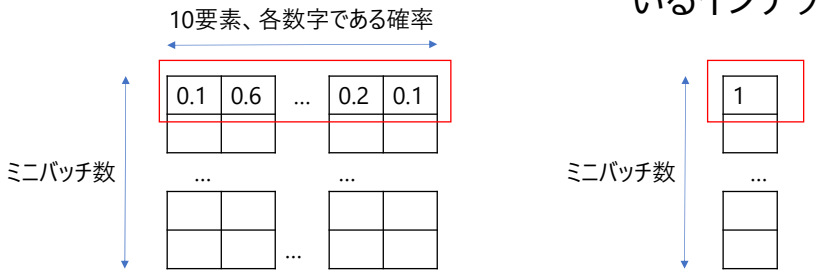
- object.__call__(self[, args...])
  - Called when the instance is "called" as a function; if this method is defined, x(arg1, arg2, ...) roughly translates to type(x).__call__(x, arg1, ...).
  - オブジェクト()　と書くと、クラス.__call__(インスタンス、引数...)　に化ける

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

F.cross_entropy(output, target)

検査するデータ

正解ラベル 1 が立っているインデックス0,..,9



赤の場合、出力は、-log(0.6)

発展：softmax の結果0..1の値でロスを計算するより、0..1の各値の対数とって-∞..0にしてから、ロス計算に回したほうが、分解能が高い。

- Net
  - forward
    - …
    - output = F.log_softmax(x, dim=1)
    - …
- Train/Test
  - …
  - loss = F.nll_loss(output, target)
  - …

発展：計算結果が毎回変わる要素を減らすため、
乱数発生器の初期値を固定してもいい。

- main
  - …
  - torch.manual_seed(args.seed)
  - …

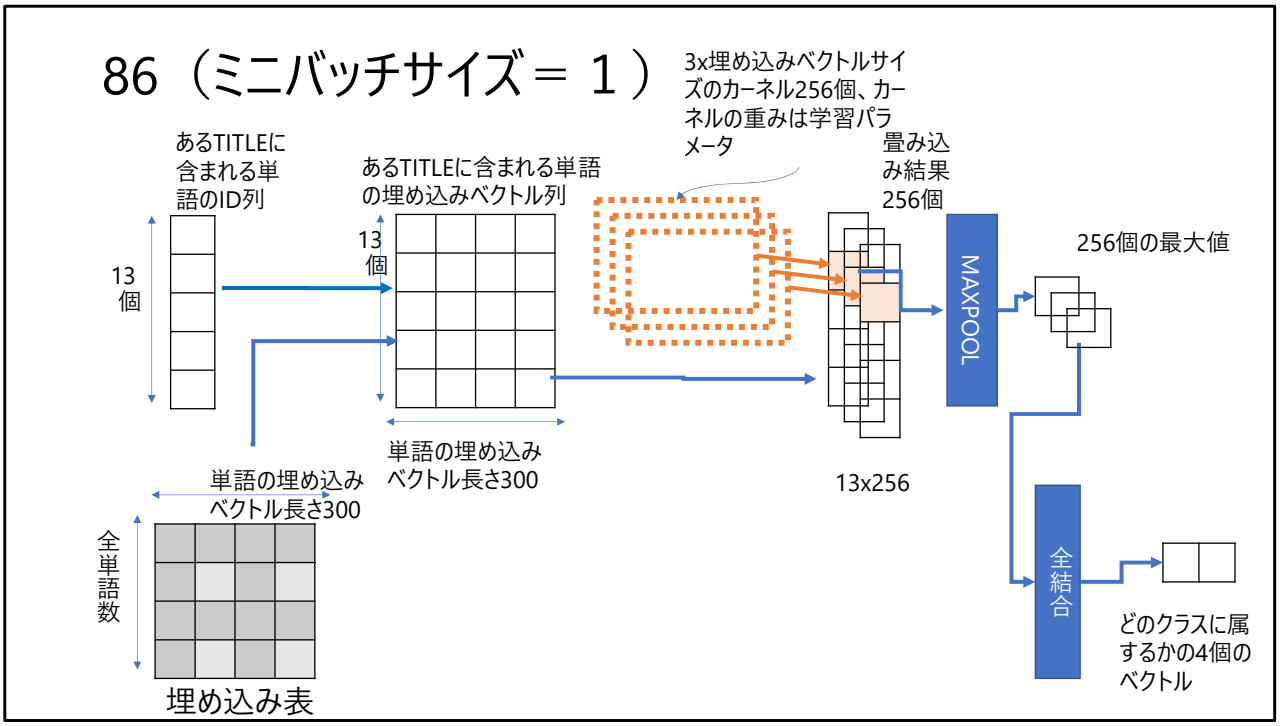
発展：過学習しないような対策

- Net
  - __init__
    - …
    - self.dropout1 = nn.Dropout2d(0.25)
    - self.dropout2 = nn.Dropout2d(0.5)
    - …
  - forward
    - …
    - x = self.dropout1(F.max_pool2d(F.relu(self.conv2(x)))
    - …
    - x = self.dropout2(F.relu(self.fc1(x)))

- モデルにランダムに欠落ノードを作ることで、常時接続した単一のモデルにはない汎用性が仕込まれるらしい。経験的な知見でしかない。

100本ノック第9章課題80,86

- 「100本ノック」の9章の課題 の80(データ準備と86/87がCNNに関する課題です。
- 「NLP、CNNRNNTransformer.ipynb」というノートをコピーして下さい。80、86回答例コードを完成し、実行ログを残してください。


86（ミニバッチサイズ＝１）
3x埋め込みベクトルサイズのカーネル256個、カーネルの重みは学習パラメータ
あるTITLEに含まれる単語のID列
13個
あるTITLEに含まれる単語の埋め込みベクトル列
13個
畳み込み結果256個
MAXPOOL
256個の最大値
単語の埋め込みベクトル長さ300
単語の埋め込みベクトル長さ300
13x256
全単語数
全結合
どのクラスに属するかの4個のベクトル
埋め込み表

確認クイズ

- 確認クイズをやってください。