MNIST with CNN

MokkeMeguru[1]

2020-02-19 Wed

[1]meguru.mokke@gmail.com

Outline

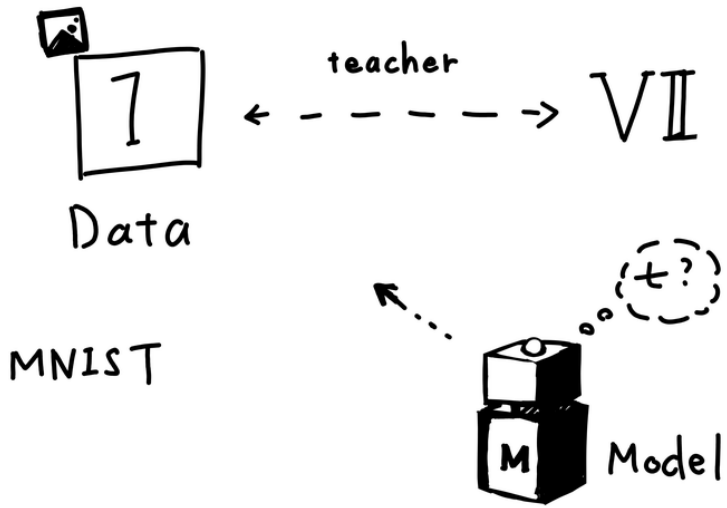
Presentaion agenda

MNIST

MNIST: 手書き文字データセット

要素

- データ
  手書き文字の画像
- モデル
  よくわからない計算グラフ
- 教師
  画像と数字 (0-9) の対応付け



Presentaion agenda

計算グラフで登場する演算要素

演算要素の例

- Dense Network
- CNN (Convolutional Neural Network)
- Activation Function
- Pooling
- (Normalization)

これらを 複数 ， うまいこと 組み合わせて計算グラフを構築する

組み合わせ数とモデルの性質の関係

- 組み合わせ数が多い
  → 複雑な計算、高性能になることもある、沢山のデータが必要
- 組み合わせ数が少ない
  → 簡単な計算、そこそこのデータでもいける

Dense Network

全結合レイヤー，線形レイヤー etc.

formula (例)

$$
\begin{aligned}
y &= Wx + b \\
\text{, where} \quad & W \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^{n} \\
& x \in \mathbb{R}^{m}, y \in \mathbb{R}^{nP}
\end{aligned}
$$

x が行列の場合も同様にして計算できる

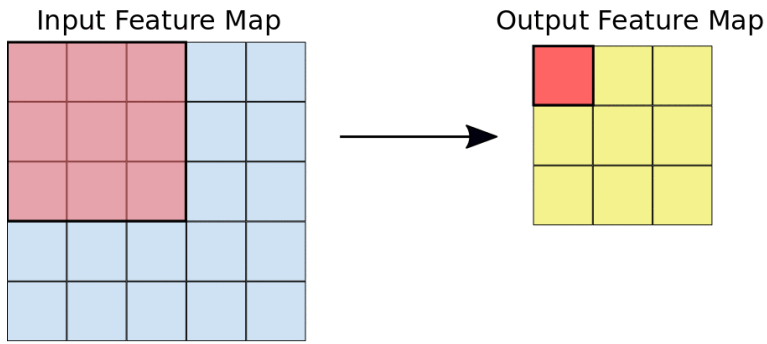
Code

```
y = tf.layers.dense(x,units=128)
# x in [64] -> y in [128]
```

CNN Convlutional Neual Network

畳み込みニューラルネットワーク (詳細は省く)

ref: Google の機械学習解説サイト



利点

- 疎結合: 大きい画像から必要な情報を抽出できる
- パラメータ共有: カーネル (薄赤) を複数の画像の位置に適用できる
  → パラメータ数の縮小
- 等価表現: 入力の違いは出力でも保持される
  → 変換前後で特徴が落ちにくい

CNN Code

- filters: カーネル (薄赤) を何枚用意するか (フィルター数)
- kernel_size: カーネルの大きさ
- padding: 端に対する処理 * 基本的には SAME で良い

```
1 y = tf.layers.conv2d(
2     x, filters=24, kernel_size=[3, 3], padding='SAME')
```

Activation Function

活性化関数
線形関数で表すことの出来ない表現を扱うための非線形関数
i.g. 例えば線形関数は XOR の関係を学習することが出来ない
$([0,1]^T, [1,0]^T \rightarrow 1, [1,1]^T [0,0]^T \rightarrow 0)$

formula (例)

$$
\begin{aligned}
y &= \mathrm{activation\_fn}(x) \\
\text{, where} \quad & x \in \mathbb{R}^x \\
& y \in \mathbb{R}^x
\end{aligned}
$$

例: ReLU

$$f(x) = max\{0, x\}$$

Code

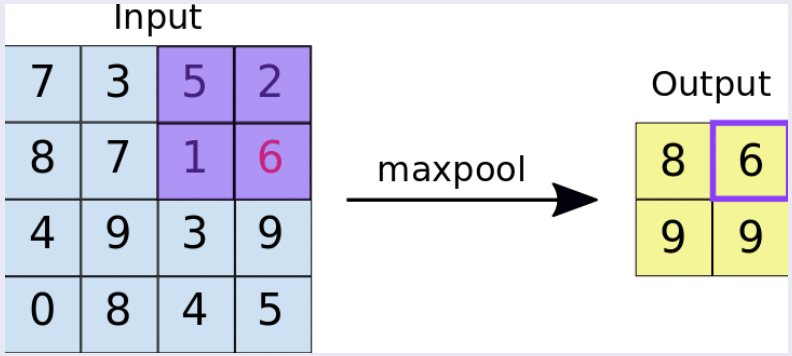
```
1 y = tf.nn.relu(x)
```

Pooling

Pooling 次元削減
i.g. Max Pooling, Average Pooling

イメージ (Max Pooling)

ref: Google の機械学習解説サイト



Code

```
pooling = tf.layers.max_pooling2d(
        pool_size=2, strides=2)
```

Presentaion agenda

CNN を用いたクラス分類のテンプレート I

CNN を用いてクラス分類する際には、
CNN->Activation Function->Pooling を 1 単位とする場合が多い

```
1  @add_arg_scope
2  def convnet(scope_name: str,
3              x: tf.Tensor,
4              filters: int,
5              kernel_size: List[int] = [5, 5],
6              pool_size: List[int] = [2, 2],
7              activation: Callable = tf.nn.relu):
8      with tf.name_scope(scope_name):
9          conv = tf.layers.conv2d(
10             x, filters=filters, kernel_size=kernel_size, padding='SAME')
11         activation = tf.nn.relu(conv)
12         pooling = tf.layers.max_pooling2d(
13             activation, pool_size=pool_size, strides=2)
14     return pooling
```

CNN を用いたクラス分類のテンプレート II

出力は 10 クラスそれぞれの 確率値 (e.g. 1 は 40%, 7 は 50%)
ベクトル → 確率ベクトル: softmax 関数を用いると良い

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{y_j}}$$

```
1 x = tf.layers.dense(x,
2                      units=10,
3                      activation=tf.nn.softmax)
```

Presentaion agenda

モデルの性能の測り方

モデルは訓練時には限られたデータでしか学習できない

$\rightarrow$ 本当に実世界で役に立つのかは不明
$\Rightarrow$ 別のデータを用いて性能を測る必要がある

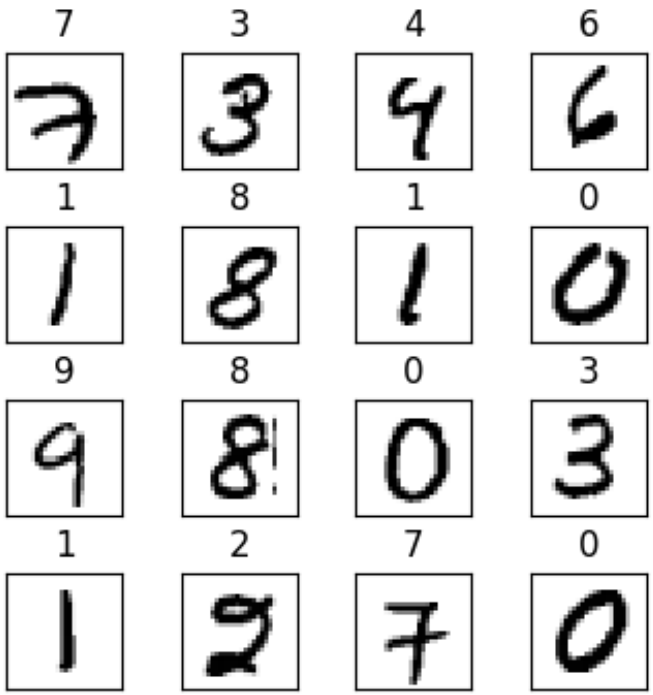
訓練データ, 検証データ, テストデータ

- 訓練データ: モデルの学習 (訓練) に使うデータ
- 検証データ: 学習中 に性能を予想するためのデータ
- テストデータ: 学習後 に性能を測るためのデータ

Presentaion agenda

課題 1

mnist 画像を python で読み込んで、表示せよ。
コードは mnist/task.py の load_mnist 関数にある。



課題 2

モデルと訓練コードを用いて、実際に訓練を行いなさい。
モデル: mnist/model.py
訓練コード: mnist/task.py

```
python task.py -t training -p tmp
```

課題 3

訓練済みモデルを用いて、性能を評価しなさい。

```
python task.py -t test -p tmp
```