12. Machine learning I

- Regression　回帰
- Overfitting　過剰適合
- Classification　分類問題
- Example: Handwritten digit recognition　例：手書き数字認識
- Support vector machines (SVMs)　サポートベクターマシン

Regression　回帰

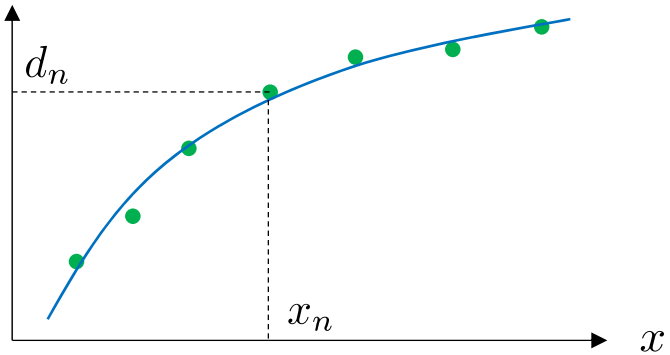
- Suppose we are given *N* pairs of a vector **x** and a scalar *d*
  *N*組のベクトル**x**とスカラー*d*が与えられたとする

$$\{\mathbf{x}_n\}(n = 1, \ldots, N) \qquad \{d_n\}(n = 1, \ldots, N)$$

- We wish to predict *d* for a new input **x**
  新しい入力**x**に対して*d*を予測したい！
  - **x**, called an independent variable, is observation used for predicting
    独立変数**x**は，予測に使用される観測値である
  - *d, called a dependent variable, is the target, or the desired value to predict*
    従属変数*d*は，予測したい値である
- Toward this goal, we consider a function that approximately satisfies
  予測のために，次の式を満たす近似関数を考える

$$y(\mathbf{x}_n) \sim d_n$$

  - You can use any arbitrary (analytical) function for y(x)
    y(x)には，任意の（解析的）関数を使用できる



Fitting polynomial functions　多項式近似

- Consider fitting a n-order polynomial func., instead of a linear func. considered earlier
  線形関数ではなく，n次多項式関数で近似することを検討する

$$y = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n$$

$$\sum_{i=1}^{N} \| y_i - (a_0 + a_1 x_i + a_2 x_i^2 + \cdots a_n x_i^n) \|^2 \rightarrow \min$$

- `polyfit` performs this　`polyfit`関数を使用する
  - E.g., You can fit a linear func. as follows, instead of using `pinv`
    例：次のように，pinvを使う代わりに線形関数を当てはめることができる

```
>> p=polyfit(x,y,1);
```

```
>> p=pinv(X)*y;
```

  - E.g., 3rd-order polynomial function
    例：３次関数近似

```
>> p=polyfit(x,y,3)
ans =
   -2.2455    3.8778   -1.3517    0.4603
```

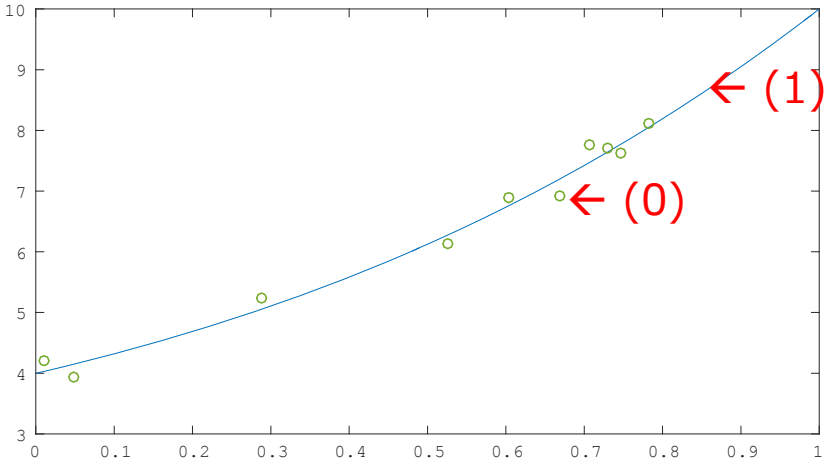
a_3　a_2　a_1　a_0

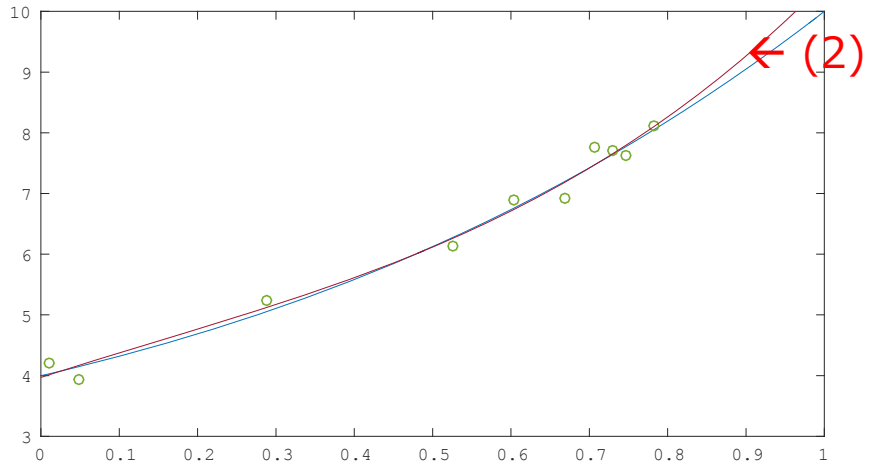
Fitting polynomial functions: an example
多項式近似：一例

```
>> x=rand(10,1);
>> p0=[1.0,2.0,3.0,4.0];
>> y=p0(1)*x.^3+p0(2)*x.^2+p0(3)*x+p0(4)+0.2*randn(10,1);
>> plot(x,y,'o') ← (0)
>>
>> hold on
>> xx=0:0.01:1;
>> yy=p0(1)*xx.^3+p0(2)*xx.^2+p0(3)*xx+p0(4);
>> plot(xx,yy) ← (1)
>>
>> p=polyfit(x,y,3);
>> plot(xx,polyval(p,xx)) ← (2)
```

Data are synthesized here for the purpose of explanation
説明のためのデータをここで合成する

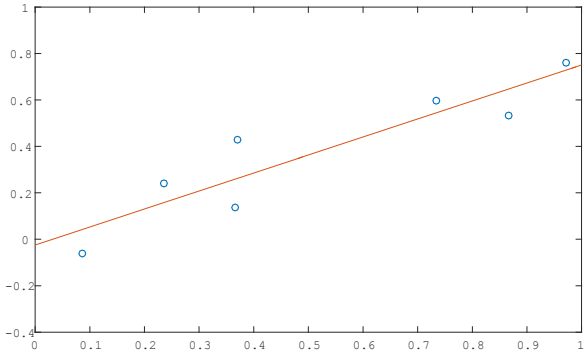
polyval :多項式の変数に値を代入して計算した結果を返してくれる関数



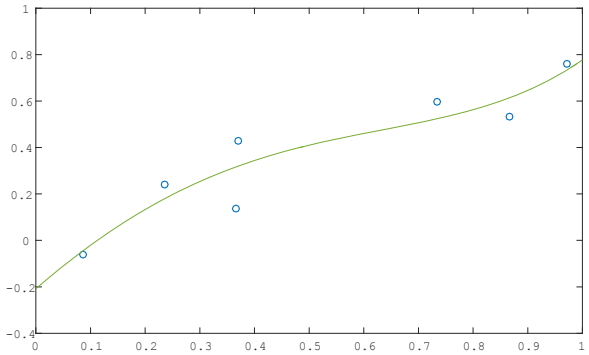


Overfitting (also called *overtraining*) 過剰適合（過学習）

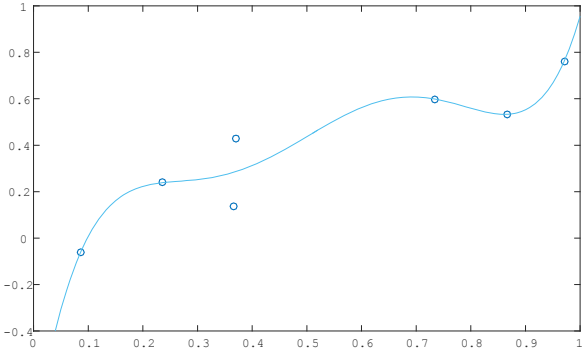
- If you fit 1st, 3rd, 5th, and 6th-order funcs to seven data points…
  1次、3次、5次、6次の関数を7つのデータ点に当てはめると…
  - Models with excessively large degrees of freedom can explain data perfectly even including their noises, which is totally meaningless!
    自由度が高すぎるモデルでは，ノイズを含めてデータを完全に説明できますが，これではまったく意味がありません



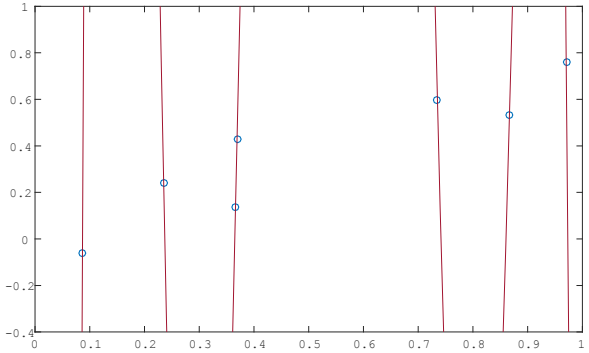
1st order (a line)



3rd-order



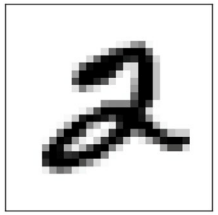
5th-order



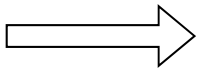
6th-order

Classification　分類

- Consider a variable **x** belonging to one of *K* classes
  K個のクラスの1つに属する変数**x**を考える
- Classification = assigning an input **x** with one of *K* class labels
  分類 = 「入力**x**にK個のクラスラベルの1つを割り当てる」こと
  - E.g., **x** is an image of a digit; we wish to answer what digit it is
    例： **x**が数字の画像で，０～９のどの数字なのか分類したい



x



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

- Supposing that *N* pairs of input **x** and its true class label *d* are given
  *N*組の入力**x**とその真のクラスラベル*d*が与えられたとする

$$\{\mathbf{x}_n\}(n = 1, \ldots, N) \qquad \{d_n\}(n = 1, \ldots, N)$$

we wish to predict which class a new input **x** belongs to

新しい入力xがどのクラスに属するかを予測したい

Example: Handwritten digit recognition
例：手書き数字の認識

- We use *MNIST*, a famous dataset of handwritten digit recognition
  手書き数字認識の有名なデータセットである*MNIST*を使用する

  `http://yann.lecun.com/exdb/mnist/`

  - Download and unzip the following file from the course page
    コースページから次のファイルをダウンロードして解凍する

    `mnist-data.zip`

  - We use the following two files today:
    以下の二つのファイルを使います

    `t10k-images-idx3-ubyte` & `t10k-labels-idx1-ubyte`

- We use *support vector machines* (*SVMs*) for classification
  分類にはサポートベクターマシン（*SVMs*）を使用する

- For this purpose, we use *liblinear*, a software library of SVM
  SVMのソフトウェアライブラリである*liblinear*を使います。

Installing *liblinear*, a software library for SVM
SVMライブラリ *liblinear* のインストール

- liblinear
  - One of the most popular libraries in machine learning created by *Machine Learning Group* at *National Taiwan University*
  - 台湾国立大学のMachine Learning Groupによって作成された，機械学習で最も人気のあるライブラリの1つ
- Download files from the URL: 以下のURLからファイルをダウンロードする
  - https://www.csie.ntu.edu.tw/~cjlin/liblinear
- Extract the downloaded file and change the current directory to `liblinear-x.xx/matlab`
  ダウンロードしたファイルを解凍し，現在のディレクトリを`liblinear-x.xx / matlab`に変更する
  - `cd /Users/xxxx/Octave/liblinear-2.11/matlab`
    ユーザ名
- Run make.m　make.mを実行する
  - `>> make`
- Add the folder to search paths
  パスをフォルダを追加する
  - `>> addpath('¥Users¥xxxx¥Octave¥liblinear-2.11¥matlab')`
    ユーザ名

Support vector machines (SVMs) (1/2)*
サポートベクターマシン（SVM）（1/2）*

- Consider two-class classification : $d_n = 1 \ \text{or} \ -1$
  2クラスの分類を考えます：$d_n = 1 \ \text{or} \ -1$

- A set of samples are given : $(\boldsymbol{x}_1, d_1), (\boldsymbol{x}_2, d_2), \cdots, (\boldsymbol{x}_N, d_N)$
  サンプルのセットが与えられている：$(\boldsymbol{x}_1, d_1), (\boldsymbol{x}_2, d_2), \cdots, (\boldsymbol{x}_N, d_N)$
- We employ the following method for classification:
  分類には以下の方法を採る

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } u(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

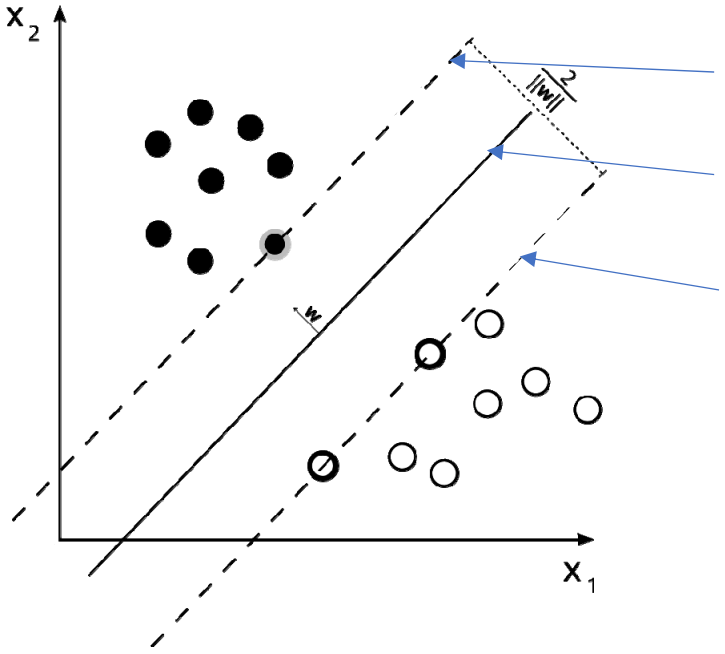
where $u(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_I x_I = w_0 + \mathbf{w}^\top \mathbf{x}$

- **w**, called weights, is a parameter to be determined
  重みと呼ばれる**w**は，決定されるパラメータ
- Consider determining w as follows:
  以下のようにwを決定する
  - Known as a *hard-margin SVM*　ハードマージン*SVM*として知られる方法

Minimize $\|\mathbf{w}\| \quad \text{subject to} \quad d_n(w_0 + \mathbf{w}^\top \mathbf{x}) \geq 1$

サポートベクターマシン（SVM）（2/2）*

- データポイントを2つの対応するクラスに正しく分割する2つの平行平面（平面間の距離を最大とする）を考える
  - 簡単のため，ここではデータ点を平面で分離できると仮定する（線形分離可能）
- 次に，パラメータw0と**w**を使い，2つの平行平面のちょうど中間にある平行平面を選択する
  - なぜこうするの？ → 新しい入力**x**を正しく分類するため，最も近いデータポイントまでの距離が最大の平面を選択することでマージンを最大にする



$$w_0 + \mathbf{w}^\top \mathbf{x} = 1$$

$$w_0 + \mathbf{w}^\top \mathbf{x} = 0$$

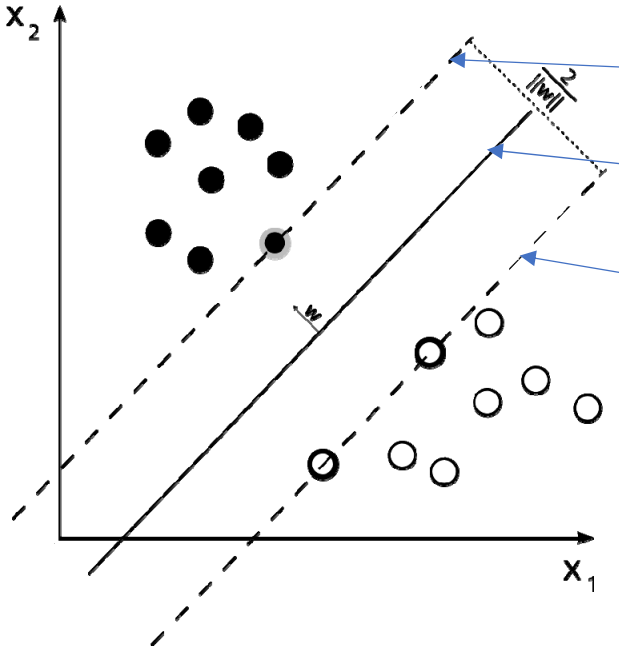
$$w_0 + \mathbf{w}^\top \mathbf{x} = -1$$

&

Minimize $\|\mathbf{w}\|$ subject to $d_n(w_0 + \mathbf{w}^\top \mathbf{x}) \geq 1$

Support vector machines (SVMs) (2/2)*

- We consider two parallel planes separating data points correctly into two corresponding classes that have the maximum distance
  - For simplicity we assume here that the data points can be separated by a plane (called *linearly separable*)
- We then choose the parallel plane in the exact middle of the two parallel plane; we use its parameters w_0 and **w**
  - Why do we do this? → It will be safe to choose the plane having the maximum distances to the nearest data points for the purpose of classifying new inputs **x**'s correctly



$$\Updownarrow$$

$$w_0 + \mathbf{w}^\top \mathbf{x} = 1$$

$$w_0 + \mathbf{w}^\top \mathbf{x} = 0$$

$$w_0 + \mathbf{w}^\top \mathbf{x} = -1$$

&

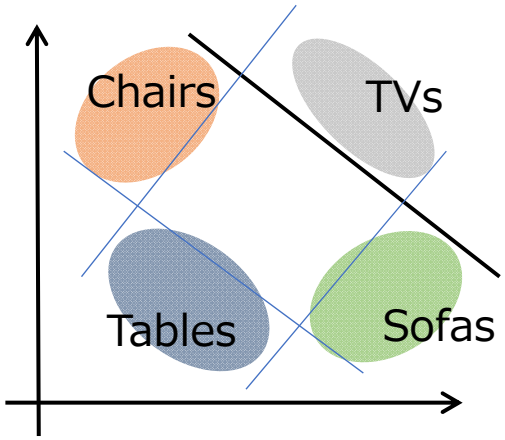
Minimize $\|\mathbf{w}\|$ subject to $d_n(w_0 + \mathbf{w}^\top \mathbf{x}) \geq 1$

Classification of multiple classes*
多クラス分類

- Two-class classifier is trained for each class to distinguish it from the others
  2クラス分類器は，他のクラスと区別するために各クラスごとに訓練されています
  - Called the *one-versus-the-rest* classifier
    1対他分類器と呼ばれる

1. k^{th} model $y_k(x)$ is trained to classify class k and other classes
   k番目のモデル$y_k(x)$はクラスkと他のクラスを分類するように訓練されています
2. Regarding the output of each model as *score* of the model, we classify an input sample to the class with the largest score
   各モデルの出力をモデルのスコアとして、入力サンプルを最大のスコアを持つクラスに分類します

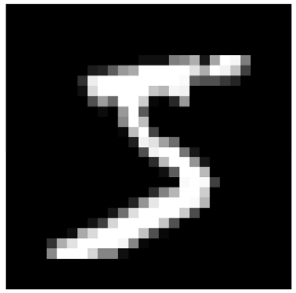
$$\underset{k}{\operatorname{argmax}}\, y_k(\mathbf{x})$$



MNISTファイルからのデータ読み込み

- Octaveへの画像の読み込み:
  - ファイル‘ test-images-idx3-ubyte ‘には、28 x 28ピクセルの10,000個の画像が含まれている
  - 最初の4つの整数（32ビット）をスキップして残りの数値データを`data`という名前の変数にロードします
  - 画像を表示するには、まず画像データを適切なサイズのテンソルに変形し、`imshow(matrix, [brightness_min, brightness_max])`を使用します。

```
>> fid=fopen('t10k-images-idx3-ubyte','r','b');
>> fread(fid,4,'int32')
>> data=fread(fid,[28*28,10000],'uint8');
>> fclose(fid);
>> img=reshape(data,28,28,10000);
>> imshow(img(:,:,1)',[0,255])
>> imshow(img(:,:,100)',[0,255])
```



- Octaveへのラベルの読み込み:
  - ファイル‘ test-labels-idx1-ubyte ‘には、画像のラベルが同じ順序で含まれている
  - 最初の2つの整数（32ビット）をスキップして、残りの整数を`label`という名前の変数にロードします

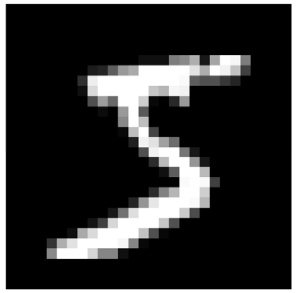
```
>> fid=fopen('t10k-labels-idx1-ubyte','r','b');
>> fread(fid,2,'int32')
>> label=fread(fid,10000,'uint8');
```

Check the contents of this variable
この変数の中身を確認！

Reading data from MNIST files

- Loading images to Octave:
  - File ‘test-images-idx3-ubyte’ contains 10,000 images of 28x28 pixels
  - Skip the first four integers (32bits) and load the remaining numerical data into a variable named `data`
  - To display images, first reshape the image data into a tensor of appropriate size and use `imshow(matrix, [brightness_min, brightness_max])`

```
>> fid=fopen('t10k-images-idx3-ubyte','r','b');
>> fread(fid,4,'int32')
>> data=fread(fid,[28*28,10000],'uint8');
>> fclose(fid);
>> img=reshape(data,28,28,10000);
>> imshow(img(:,:,1)',[0,255])
>> imshow(img(:,:,100)',[0,255])
```



- Loading labels to Octave:
  - File ‘test-labels-idx1-ubyte’ contains labels of the images in the same order
  - Skip the first two integers (32bits) and load the remaining integers into a variable named `label`

```
>> fid=fopen('t10k-labels-idx1-ubyte','r','b');
>> fread(fid,2,'int32')
>> label=fread(fid,10000,'uint8');
```

Check the contents of this variable

分類器のトレーニングとテスト

- データから5,000サンプル（画像）を使用して分類器を訓練する
  - インデックス1，…，5000のサンプルを使用してモデル（SVM）を学習させる

```
>> tr_label = label(1:5000);
>> tr_data = data(:,1:5000);
>> model = train(tr_label,sparse(tr_data)');
...
Objective value = -0.081903
nSV = 910
```

Status of training, which you can ignore (as long as the training went well)
トレーニングのステータス（トレーニングがうまくいっていれば無視できる）

- 残りのサンプルを使用して分類器の性能を評価する
  - インデックス5001，…，6000のサンプルを使用してモデルをテストする

```
>> te_label = label(5001:6000);
>> te_data = data(:,5001:6000);
>> pred_label = predict(te_label,sparse(te_data)',model)
Accuracy = 84.6% (846/1000)
pred_label =
   2
   3
...
```

Classification accuracy for the input 1,000 samples is shown
入力1,000サンプルの分類精度が表示されます

1,000サンプルの予測ラベル； 数字が実際の数字に対応していないことに注意！
これらの番号はmodel.Label （数字の本当のラベルを格納する）のインデックスに対応する.

Training and testing a classifier

- Train a classifier using, say, 5,000 samples (images) from the data
  - Train a model (SVM) using samples with indices 1,⋯,5000:

```
>> tr_label = label(1:5000);
>> tr_data = data(:,1:5000);
>> model = train(tr_label,sparse(tr_data)');
...
Objective value = -0.081903
nSV = 910
```

Status of training, which you can ignore (as long as the training went well)

- Evaluate the performance of the classifier using the remaining samples
  - Test the model using samples with indices 5001,⋯,6000:

```
>> te_label = label(5001:6000);
>> te_data = data(:,5001:6000);
>> pred_label = predict(te_label,sparse(te_data)',model)
Accuracy = 84.6% (846/1000)
pred_label =
   2
   3
...
```

Classification accuracy for the input 1,000 samples is shown

Predicted labels for the 1,000 samples; note that the numbers do not correspond to the true digits; these numbers correspond to the indices of model.Label, which stores the true labels of digits

Visualization of weights*　重みの可視化

- `predict` performs the following computation
  `predict` 関数は以下の計算をする

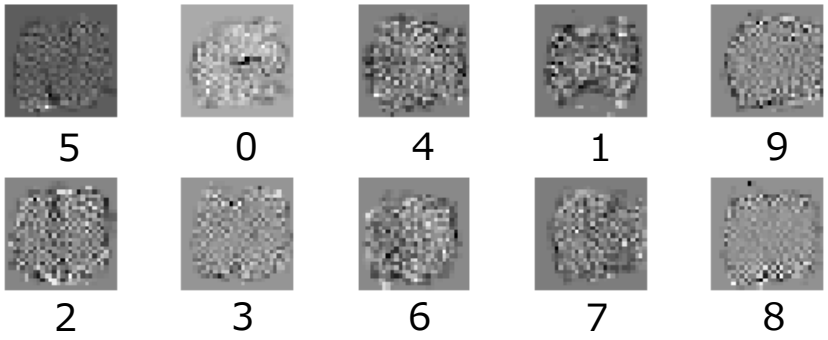
```
>> for i=1:10,model.w(i,:)*reshape(te_data(:,4),28*28,1)+model.bias,end
ans = -5.3081
...
...
ans = -17.245
ans =  2.5717
...
```

```
>> te_label(4)
ans =  6
>> model.Label
ans =

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

- Visualize the trained weights as images
  訓練された重みを画像として視覚化
  - Can you tell where in the image the model looks at to classify each digit?
    モデルが各数字を分類するために画像のどこを見ているのかわかりますか？

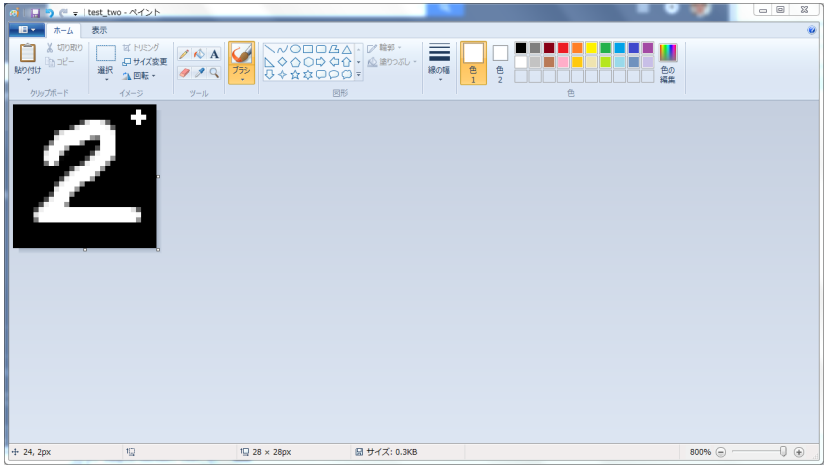
```
>> figure
>> for i=1:10,subplot(2,5,i),imshow(reshape(model.w(i,:),28,28),[min(model.w(i,:)),max(model.w(i,:))]),end
```

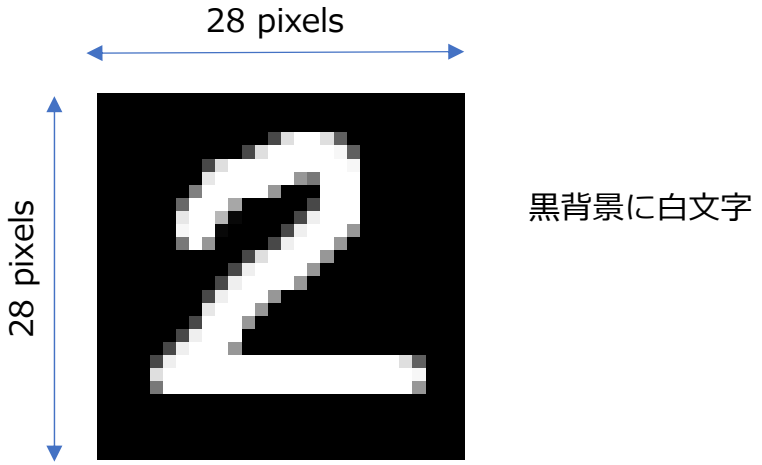


The order of weights is specified by `model.Label`
重みの順序は`model.Label`によって指定されます

Exercise 12.1 (手書きの数字をモデルに認識させる)

- 28×28ピクセルの画像を作成し，その中にお気に入りの数字を描き、それをファイルに保存する．たとえば，ペイントを使用する．



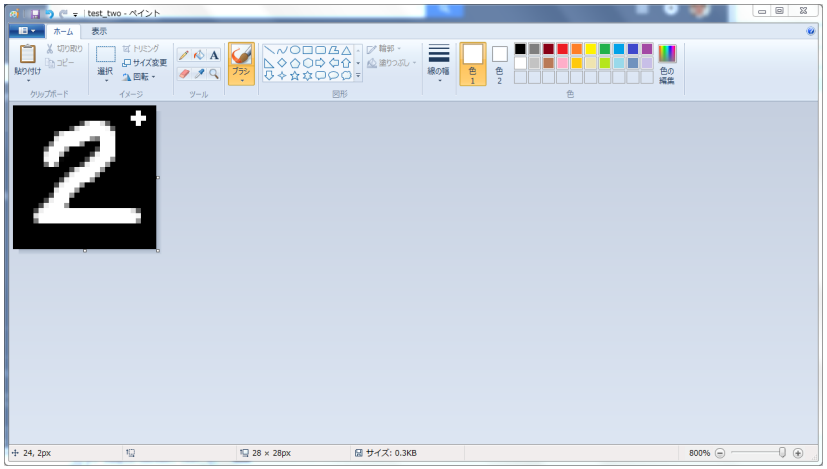


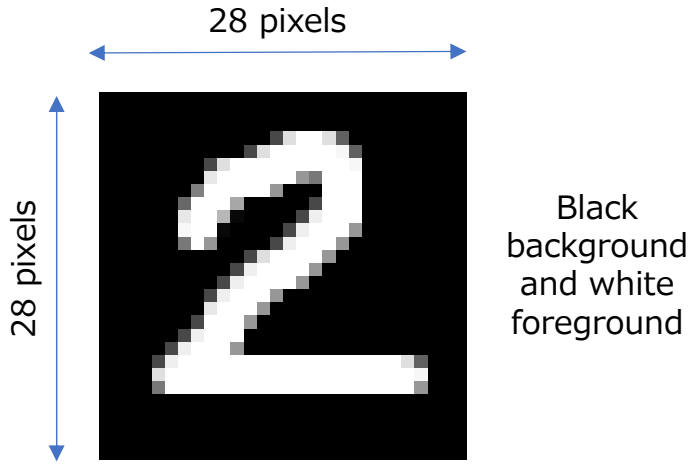
- まず訓練したモデルを使用して数字を認識する
  - 正しい結果が得られない場合は，数字を再描画してもう一度予測してください

```
>> sample = imread('a_number_I_wrote.png');
>> sample = mean(sample,3);  カラー画像の場合は，画像をグレースケールに変換する
          真ラベル
>> predict([2], sparse(reshape(sample',1,28*28)), model)
Accuracy = 100% (1/1)
ans =  2
       予測されたラベル（正解）
```

Exercise 12.1 (Make the model recognize your handwritten digit)

- Create an image of 28x28 pixels, draw your favorite digit in it, and save it to a file, by using, say, Paint of Windows





- Use the model we trained earlier to recognize the digit
  - If the correct result is not obtained, redraw a digit and predict again

```
>> sample = imread('a_number_I_wrote.png');
>> sample = mean(sample,3);   Convert your image into grayscale if it is a color image
                 True label
>> predict([2], sparse(reshape(sample',1,28*28)), model)
Accuracy = 100% (1/1)
ans =  2
          Predicted label; this is correct!
```