12. Machine learning I

- Regression
- Overfitting
- Classification
- Example: Handwritten digit recognition
- Support vector machines (SVMs)

Regression

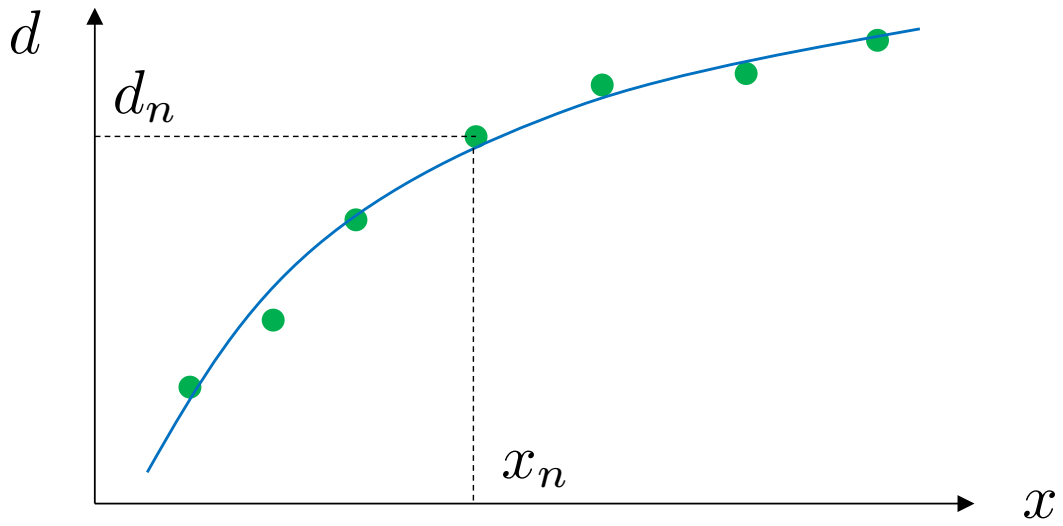
- Suppose we are given *N* pairs of a vector **x** and a scalar *d*

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

- We wish to predict *d* for a new input **x**
  - **x**, called an independent variable, is observation used for predicting
  - *d, called a dependent variable, is the target, or the desired value to predict*
- Toward this goal, we consider a function that approximately satisfies

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

  - You can use any arbitrary (analytical) function for y(x)



Fitting polynomial functions

- Consider fitting a n-order polynomial func., instead of a linear func. considered earlier

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

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

- `polyfit` performs this
  - E.g., You can fit a linear func. as follows, instead of using `pinv`

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

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

  - E.g., 3rd-order polynomial function

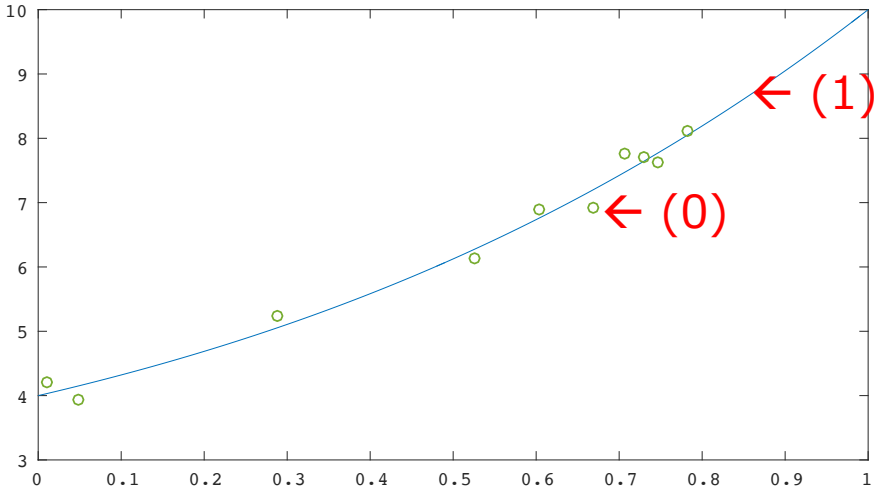
```
>> p=polyfit(x,y,3)
ans =
   -2.2455    3.8778   -1.3517    0.4603
```

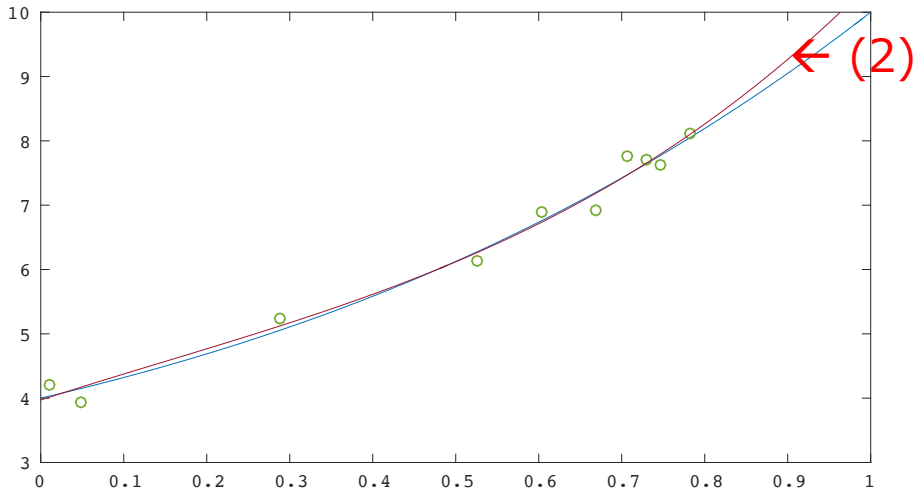
a_3 a_2 a_1 a_0

Fitting polynomial functions: an example

```
>> 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)
```

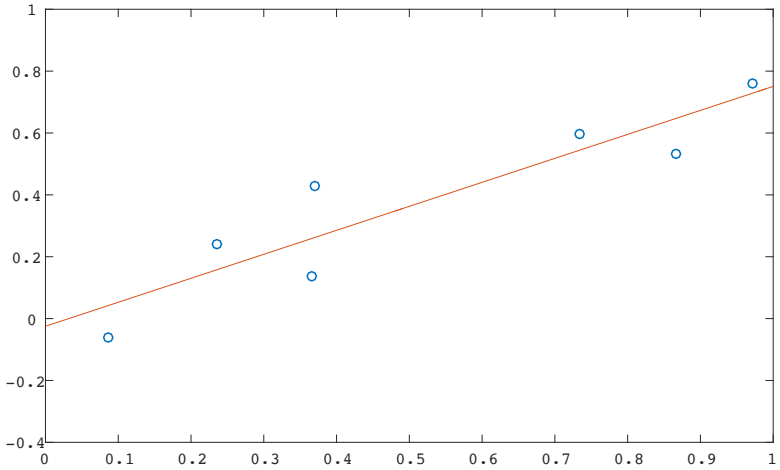
Data are synthesized here for the purpose of explanation



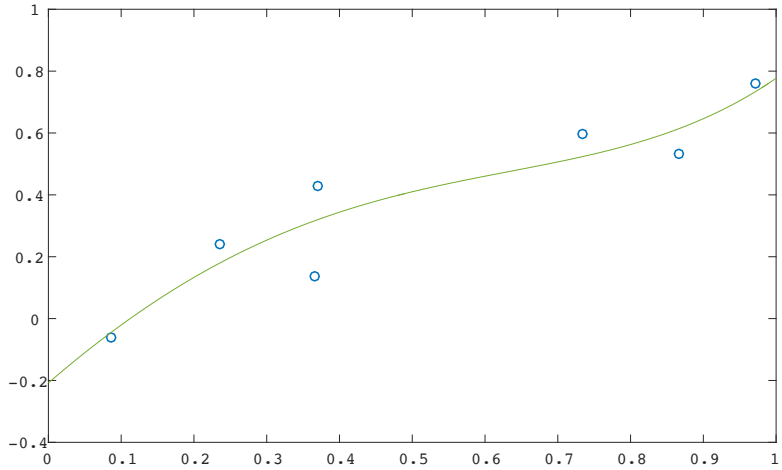


Overfitting (also called *overtraining*)

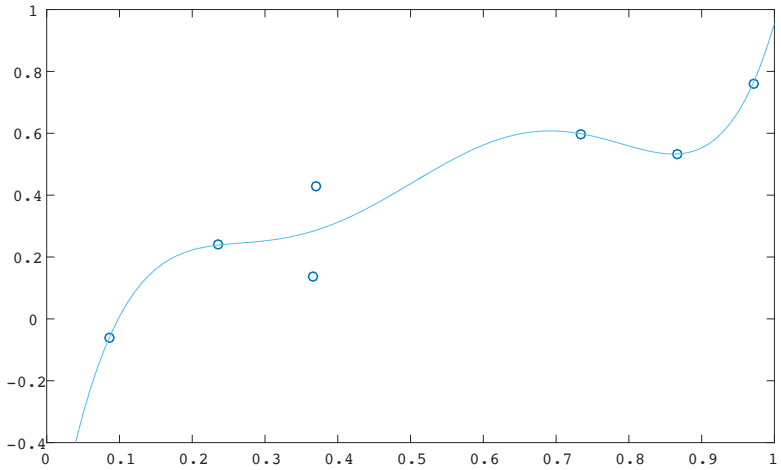
- If you fit 1st, 3rd, 5th, and 6th-order funcs to seven data points…
  - Models with excessively large degrees of freedom can explain data perfectly even including their noises, which is totally meaningless!



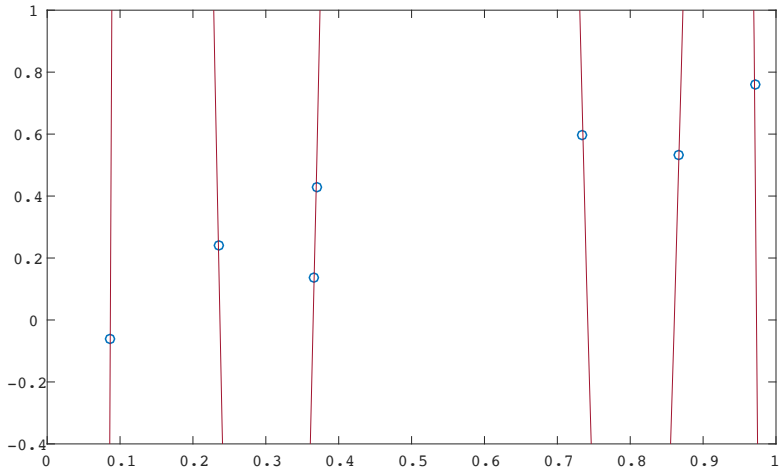
1st order (a line)



3rd-order



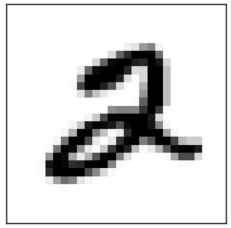
5th-order



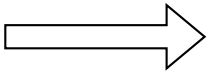
6th-order

Classification

- Consider a variable **x** belonging to one of *K* classes
- Classification = assigning an input **x** with one of *K* class labels
  - E.g., **x** is an image of a digit; we wish to answer what digit it is



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

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

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

Example: Handwritten digit recognition

- We use *MNIST*, a famous dataset of handwritten digit recognition

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

  - Download and unzip the following file from the course page

    `mnist-data.zip`

  - We use the following two files today:

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

- We use *support vector machines* (*SVMs*) for classification
- For this purpose, we use *liblinear*, a software library of SVM

Installing *liblinear*, a software library for SVM

- liblinear
  - One of the most popular libraries in machine learning created by *Machine Learning Group* at *National Taiwan University*
- Download files from the URL:
  - https://www.csie.ntu.edu.tw/~cjlin/liblinear
- Extract the downloaded file and change the current directory to `liblinear-x.xx/matlab`
  - `cd /Users/xxxx/Octave/liblinear-2.11/matlab`
- Run make.m
  - `>> make`
- Add the folder to search paths
  - `>> addpath('/Users/xxxx/Octave/liblinear-2.11/matlab')`

Support vector machines (SVMs) (1/2)*

- Consider two-class classification : $d_n = 1 \ \text{or} \ -1$
- A set of samples are given : $(\boldsymbol{x}_1, d_1), (\boldsymbol{x}_2, d_2), \cdots, (\boldsymbol{x}_N, d_N)$
- We employ the following method for classification:

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

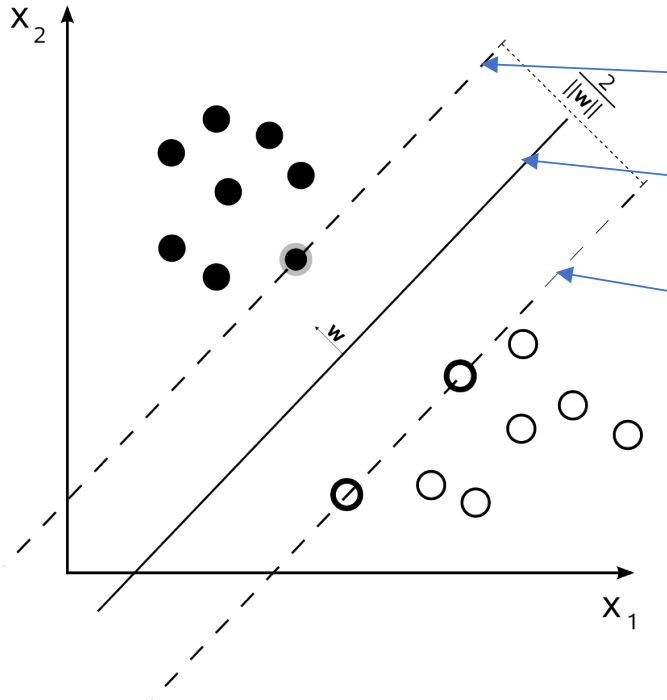
where $u(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_I x_I = w_0 + \mathbf{w}^\top \mathbf{x}$

- **w**, called weights, is a parameter to be determined
- Consider determining w as follows:
  - Known as a *hard-margin SVM*

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

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



$$\Updownarrow$$

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

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

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

&

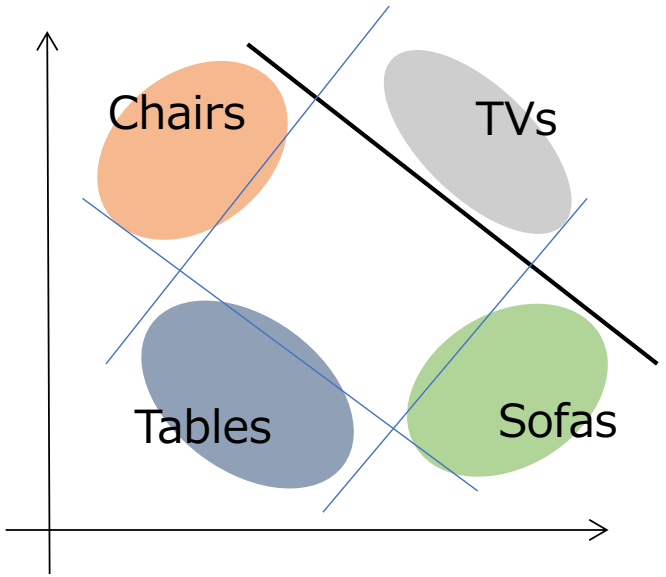
Minimize $\|\mathbf{w}\|$ subject to $d_n(w_0 + \mathbf{w}^\top \mathbf{x}) \geq 1$

Classification of multiple classes*

- Two-class classifier is trained for each class to distinguish it from the others
  - Called the *one-versus-the-rest* classifier

1. k^{th} model $y_k(x)$ is trained to classify class k and other classes
2. Regarding the output of each model as *score* of the model, we classify an input sample to the class with the largest score

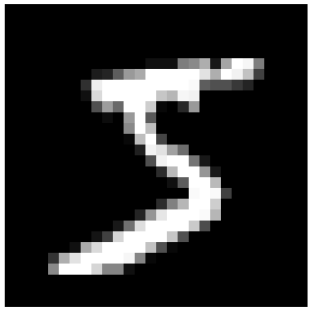
$$\underset{k}{\operatorname{argmax}}\, y_k(\mathbf{x})$$



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

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*

- `predict` performs the following computation

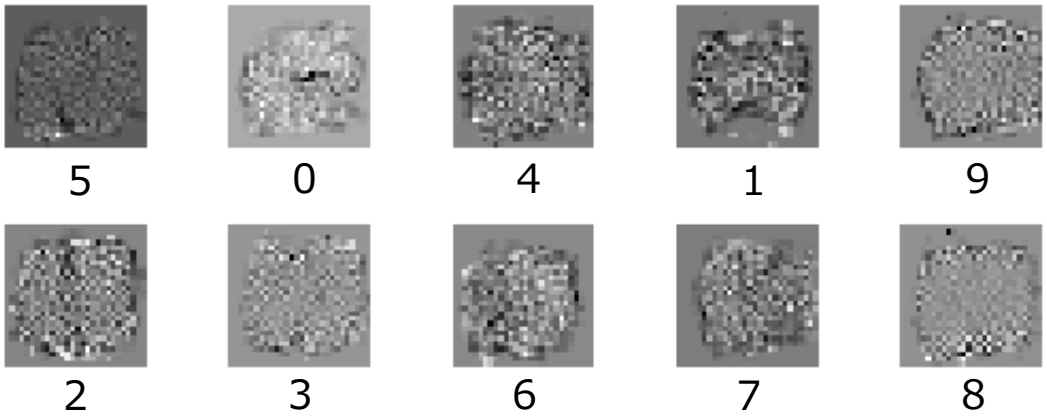
```
>> for i=1:10,model.w(i,:)*reshape(te_data(:,4),28*28,1)+model.bias,end
ans = -5.3081
...
...
ans = -17.245
ans =  2.5717
...
```

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

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

- Visualize the trained weights as images
  - Can you tell where in the image the model looks at to classify each digit?

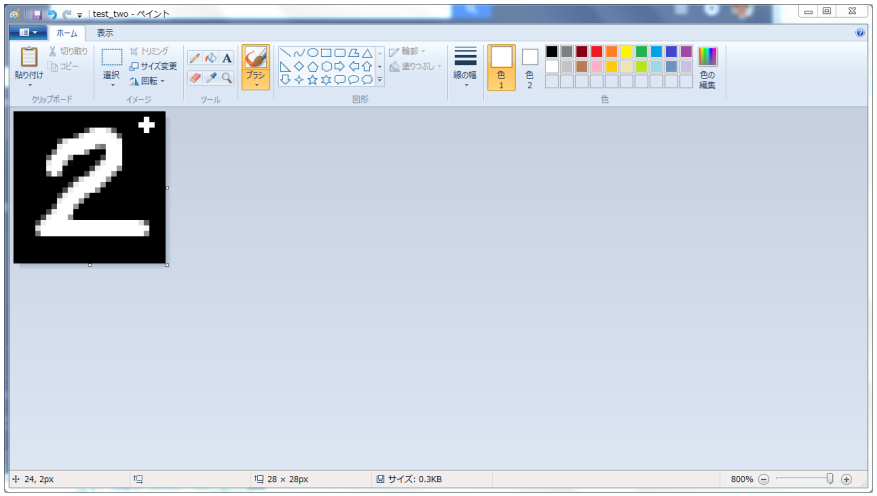
```
>> figure
>> for i=1:10,subplot(2,5,i),imshow(reshape(model.w(i,:),28,28),[min(model.w(i,:)),max(model.w(i,:))]),end
```

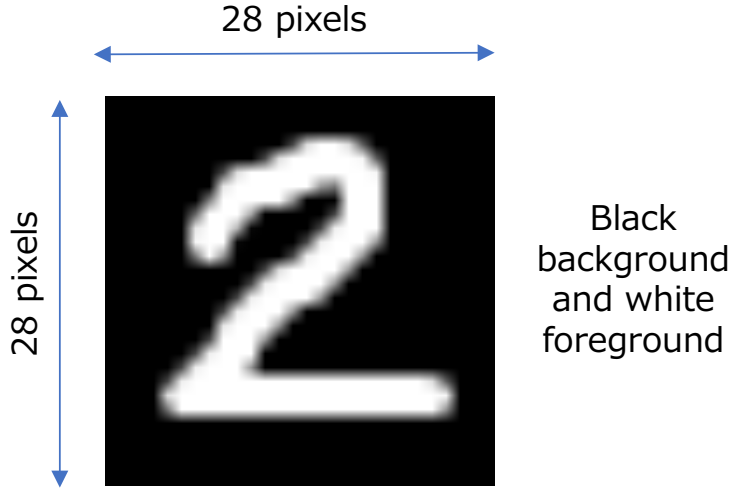


The order of weights is specified by `model.Label`

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!
```