3rd day, Afternoon session

Very beginning of Neural-Network

R-CCS HPC Summer School 2021
Lecture: Dr. Toshiyuki Imamura, RIKEN R-CCS
Assistant: Dr. Takeshi Terao and Dr. Takaaki Fukai, RIKEN R-CCS

What do you image AI or deep-machine-learning?

Honestly, …

I am not an AI expert.

It is a good experience to learn it from non-AI persons.

Textbook:

N. Buduma, Fundamentals of Deep Learning, designing next-generation machine intelligence algorithms, O’Reiley

https://www.oreilly.com/library/view/fundamentals-of-deep/9781491925607/

AI

- AI (Artificial Intelligence) = Intelligent Machine
- Q. Can we emulate human brain on our computer system?

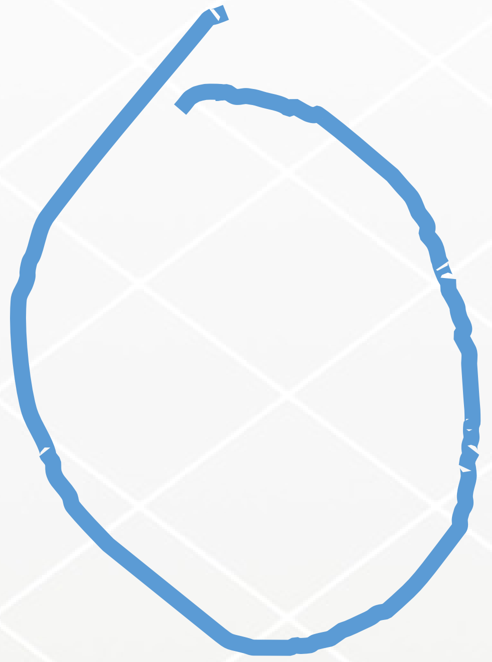
Brain is, inherently, what makes us intelligent.

- Dream of building intelligent machine with brains like ours is …
- We have to develop a radically difference programming a computer using techniques largely developed over the past decade.

The limits of Traditional ways

- **Traditional Computer program**
  - can do the trick for determined or non-vague issues.

  - How to recognize a messy 0 from a 6 ?
  - How do we write a program for that ?



← 6 or 0 ?

You can document the rules one after another. For example, points, angles, and rounds (radii) of the writing/drawing character.

Very famous MNIST handwritten letter test

Different approach

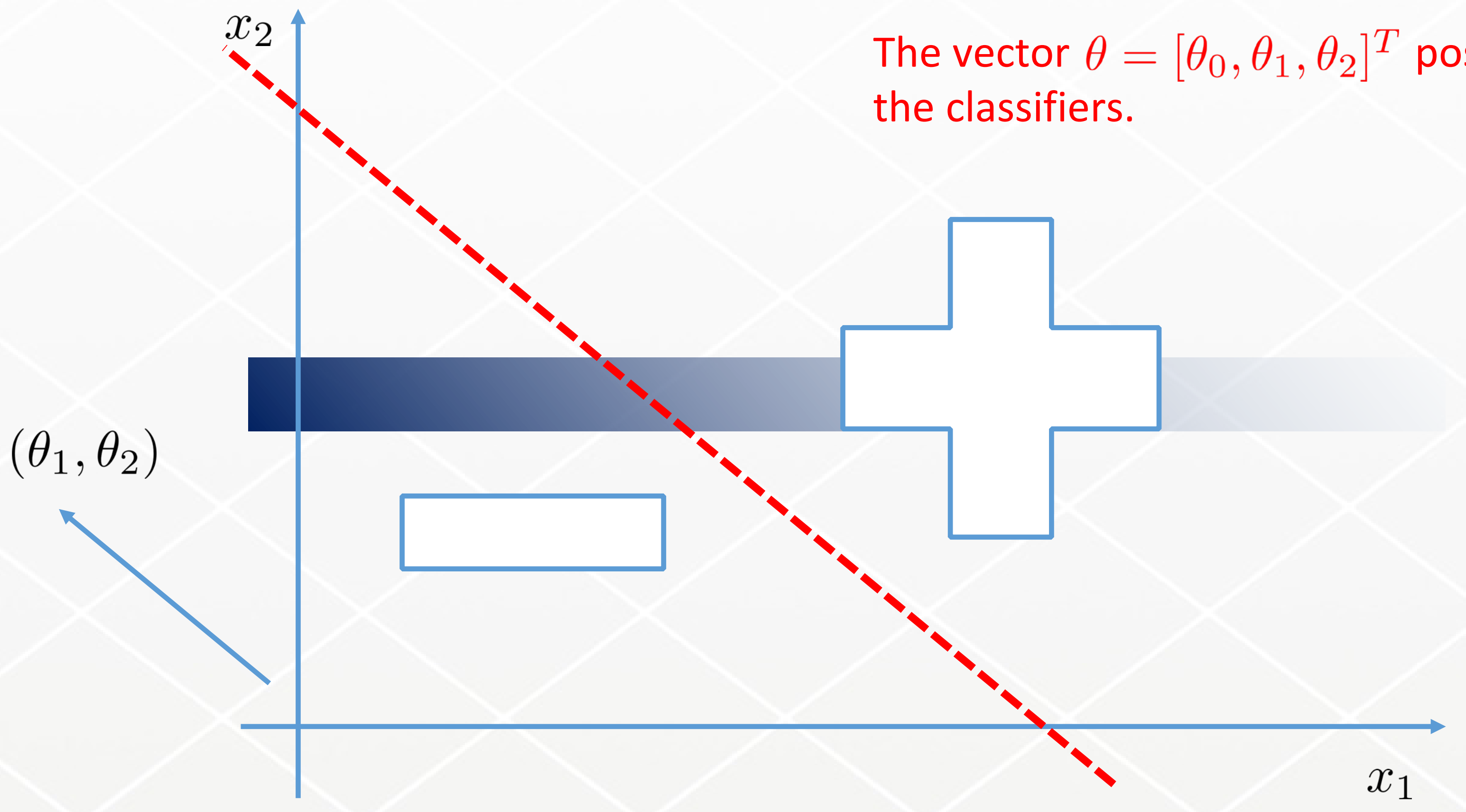
- **As human learn a lot of things at school,**
  - how to multiply numbers, solve equations, derivatives, and further
  - The things we find most natural, are learned by EXAMPLES, not by formula.
  - Deep Learning = subset of a more general field of AI, machine learning.

Linear Perceptron

$$h(\boldsymbol{x},\theta) = \begin{cases} -1 & \text{if} \quad \boldsymbol{x}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \theta_0 < 0 \\ +1 & \text{if} \quad \boldsymbol{x}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \theta_0 \geq 0 \end{cases}$$

Linear Perceptron

$$h(\boldsymbol{x}, \theta) = \begin{cases} -1 & \text{if} \quad \boldsymbol{x}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \theta_0 < 0 \\ +1 & \text{if} \quad \boldsymbol{x}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + \theta_0 \geq 0 \end{cases}$$

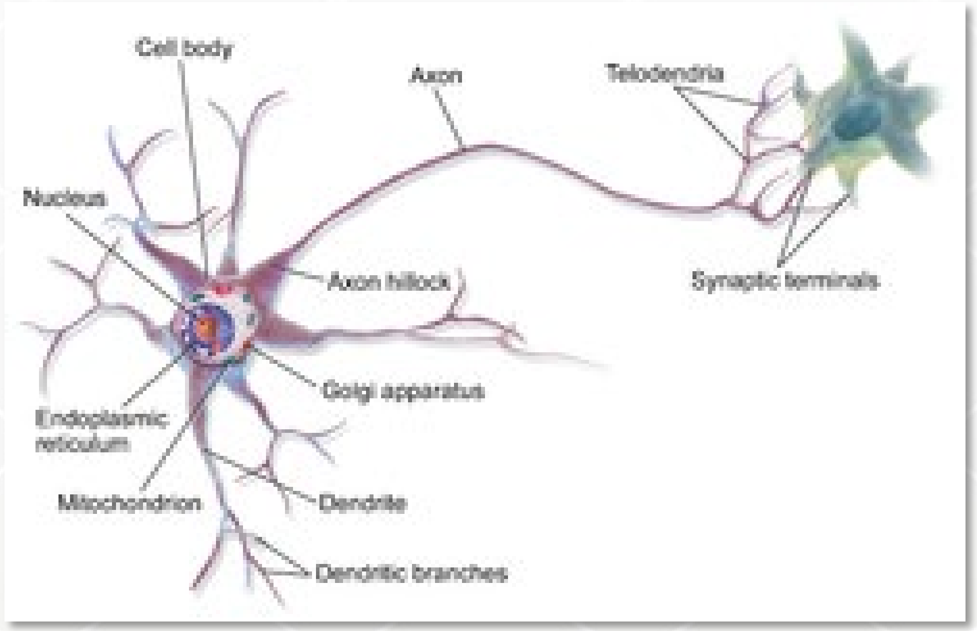


The vector $\theta = [\theta_0, \theta_1, \theta_2]^T$ positions the classifiers.

The Neuron

- ***The foundational unit of the human brain is the neuron. A tiny piece of the brain, about the size of grain of rice, contains over 10,000 neurons, each of which forms an average of 6,000 connections with other neurons. It's this massive biological network that enables us to experience the world around us.***

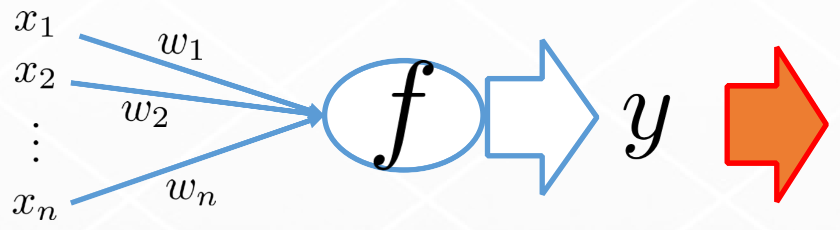
N. Buduma, Fundamentals of Deep Learning, designing next-generation machine intelligence algorithms



https://en.wikipedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png

Neurons 🡐 Linear Perceptrons

- **Modeling a neuron as a network [1943, Warren, McCulloch, Pitts], $y = f(z), z = (w, x) + b$.**

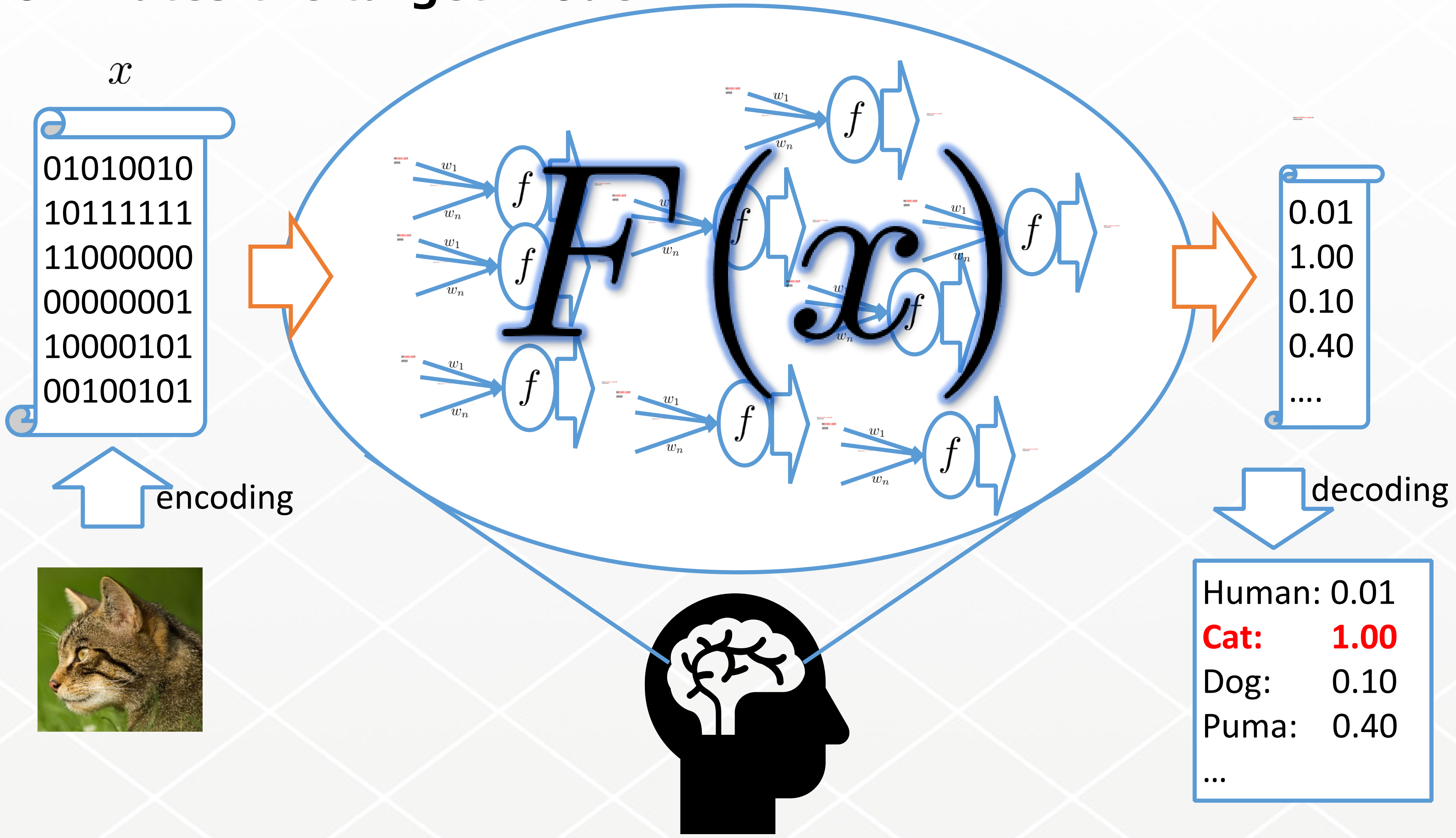


$$f(z) = \begin{cases} -1 & \text{if} \quad z < 0 \\ +1 & \text{if} \quad z \geq 0 \end{cases}$$

- **Neurons in the human brain are layered.**
  - The human cerebral cortex ➔ six layers.
  - The simplest network is called feed-forward network
  - Linear Perceptron has easiness but limitation in the capability to express the hidden layers, which are sandwiched between the first input and the last output layers.
  - We need to employ some sort of non-linearity in order to learn the complex relationships.

Short Summary of Neural Network (NN)

- We need to **find a very complex function $F(x)$, or its parameter set** that approximates the target model.

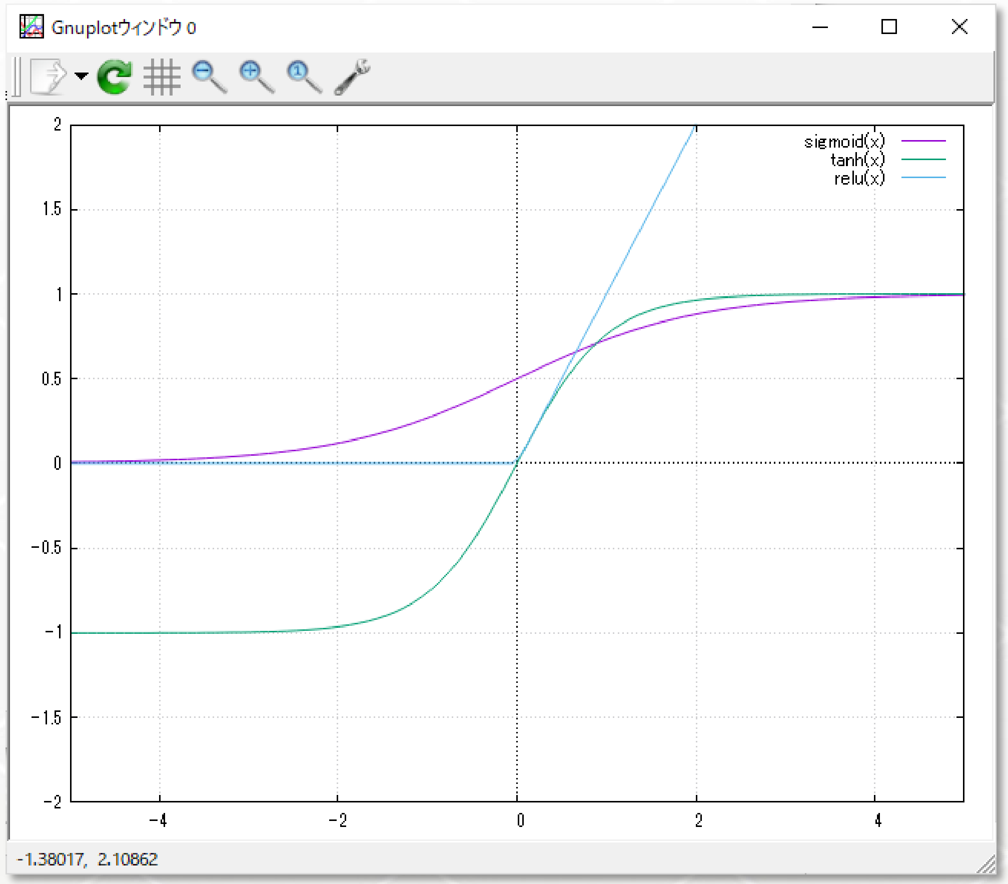


Short break

- **Which function should we use neuron $f(z)$ (namely, activation function) ?**

Sigmoid	Tanh
$\frac{1}{1+e^{-z}}$	$\tanh z$

ReLU	softmax
$\max(0, z)$	$\frac{e^{z_j}}{\sum e^{z_j}}$



Training Feed-Forward network

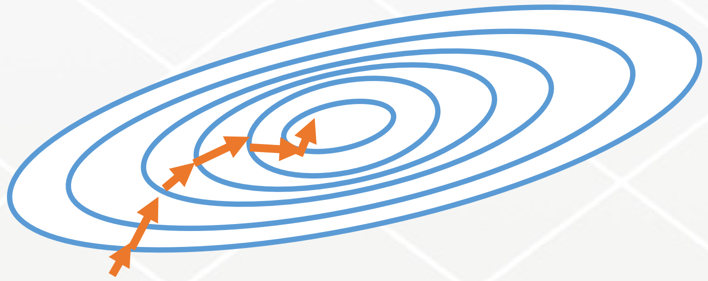
- **Supposed:**
  - We have a large number set of training examples.
  - Also, we can calculate the return value/vector of the neural-network when we input a training set.
- **Definition of Error:**
  - By SSE (or MSE) function as follows

$$E = \frac{1}{2}\sum_i \left(t^{(i)} - y^{(i)}\right)^2$$

$t^{(i)}$: i-th training dataset

$y^{(i)}$: corresponding output value

- **Gradient descent direction:**
  - If we plot a contour map of the Error function,



Steep down to the descent direction

$$-\nabla E$$

We can minimize the term $\square$*, step by step*

$$\square \leftarrow \square - \eta\frac{\partial E}{\partial \square} = \square - \Delta\square$$

Training Feed-Forward network

- **How to Change weight parameters:**
  - Calculate partial derivative of the error functions with respect to each of the weights.

$$E = \frac{1}{2}\sum_i \left(t^{(i)} - y^{(i)}\right)^2$$

$$\square \leftarrow \square - \eta\frac{\partial E}{\partial \square} = \square - \Delta\square$$

$$\Delta w_k = -\epsilon\frac{\partial E}{\partial w_k} = \sum_i \epsilon\left(t^{(i)} - y^{(i)}\right)\frac{\partial y^{(i)}}{\partial w_k}$$

$$z = \sum_k w_k x_k, y = f(z) \Rightarrow \frac{\partial y}{\partial w_k} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_k} = f'(z)x_k$$

  - If we use sigmoidal neurons, $y = f(x) = 1/(1 + \exp(-z))$,

$$\Delta w_k = -\epsilon\frac{\partial E}{\partial w_k} = \sum_i \epsilon\left(t^{(i)} - y^{(i)}\right)\left(y^{(i)}(1 - y^{(i)})\right)x_k^{(i)}$$

  - By using the obtained rule for the weights, we modify all the weights step by step.

Framework of the backward calculation

- **If only output layer is considered for the Error:**

$$\boxed{\frac{\partial E}{\partial y_j}} = -(t_j - y_j)$$

- **Next, at j-th layer. We calculate how the outgoing j-th layer y_j affects incoming i-th layer (i→j)**
  - Similar to the output layer, we have following relation

$$\boxed{\frac{\partial E}{\partial y_i}} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} = \sum_j w_{ij} f'_j(z_j) \boxed{\frac{\partial E}{\partial y_j}}$$

  - Next, ∂E/∂wij is obtained by compound expressions with chain rule

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial y_j}{\partial z_j}\frac{\partial E}{\partial y_j} = y_{ij} f'(z_j) \boxed{\frac{\partial E}{\partial y_j}}$$

Back-Propagation

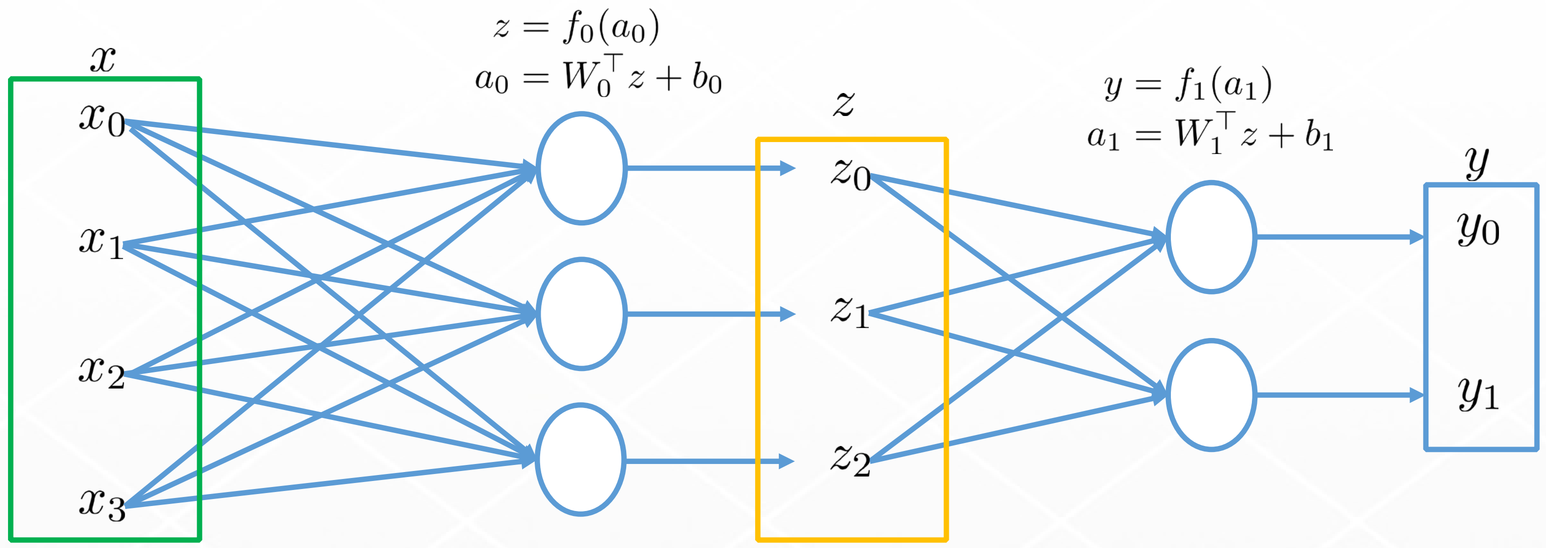
- As last slide, We know only the relation of the output data and test data.

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$

- Therefore, the method of calculating the error in the middle layer from the error in the final output layer (backward) to initial is significant, called “backpropagation”

Chain rule: $$\frac{\partial f}{\partial u} = \frac{\partial x}{\partial u}\frac{\partial f}{\partial x} + \frac{\partial y}{\partial u}\frac{\partial f}{\partial y} + \cdots$$

$$f = f(x(u, v, \cdots), y(u, v, \cdots), \cdots)$$



$$\boxed{\frac{\partial E}{\partial z_i}} = \sum_j \frac{da_{1j}}{dz_i}\frac{\partial E}{\partial a_{1j}} = \sum_j \frac{da_{1j}}{dz_i}\frac{\partial y_j}{\partial a_{1j}}\frac{\partial E}{\partial y_j} = \sum_j W_{1ji}\nabla f_{1j}(a_1)(y_j - t_j)$$

$$\Rightarrow \nabla_z E = W_1 \Delta y, \;\; \Delta y = \nabla f_1(a_1) \odot (y - t)$$

$$\boxed{\frac{\partial E}{\partial x_i}} = \sum_j \frac{da_{0j}}{dx_i}\frac{\partial E}{\partial a_{0j}} = \sum_j \frac{da_{0j}}{dx_i}\frac{\partial z_j}{\partial a_{0j}}\frac{\partial E}{\partial x_j} = \sum_j W_{0ji}\nabla f_{0j}(a_0)\frac{\partial E}{\partial z_j}$$

$$\Rightarrow \nabla_x E = W_0 \Delta z, \;\; \Delta z = \nabla f_0(a_0) \odot \nabla_z E$$

$\odot$: Hadamard product (elementwise mult.)

- **We obtained the gradient of E in terms of z and x.**

$$\nabla_z E = W_1 \Delta y, \ \Delta y = \nabla f_1(a_1) \odot (y - t)$$
$$\nabla_x E = W_0 \Delta z, \ \Delta z = \nabla f_0(a_0) \odot \nabla_z E$$

- **From them, we update weighted factors {W, b}.**

$$\Delta W_{1ij} = -\epsilon \frac{\partial E}{\partial W_{1ij}} = -\epsilon z_i \Delta y_j$$
$$\Delta W_{0ij} = -\epsilon \frac{\partial E}{\partial W_{0ij}} = -\epsilon x_i \Delta z_j$$

$$\Rightarrow$$

$$\Delta W_1 = -\epsilon z \Delta y^\top$$
$$\Delta W_0 = -\epsilon x \Delta z^\top$$

$$\Delta b_{1i} = -\epsilon \frac{\partial E}{\partial b_{1i}} = -\epsilon \Delta y_i$$
$$\Delta b_{0i} = -\epsilon \frac{\partial E}{\partial b_{0i}} = -\epsilon \Delta z_i$$

$$\Rightarrow$$

$$\Delta b_1 = -\epsilon \Delta y$$
$$\Delta b_0 = -\epsilon \Delta z$$

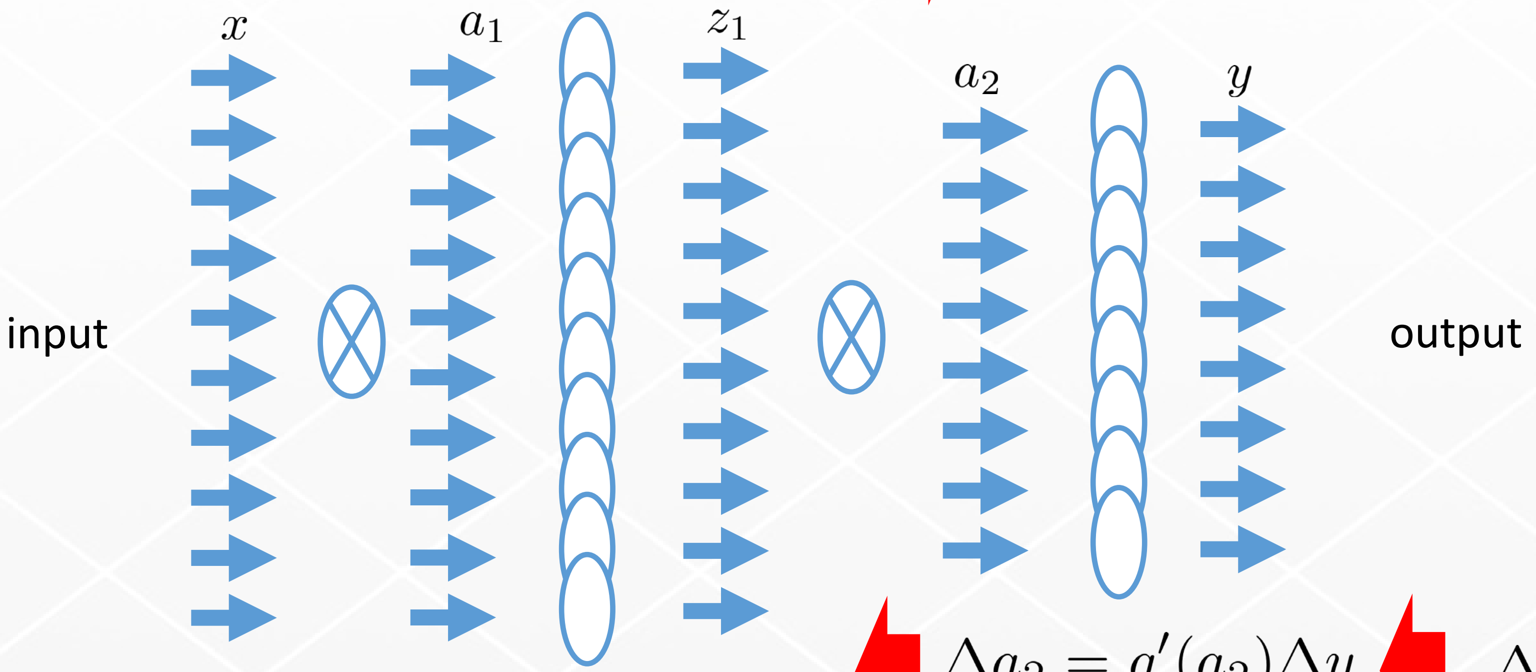
← Please confirm the equations by yourselves

Summary [Schematics of a simple case in a matrix-vector form]

Forward:

$$a_1 = W_1^\top x + b_1$$
$$z_1 = f(a_1)$$

$$a_2 = W_2^\top z_1 + b_2$$
$$y = g(a_2)$$



Backward:

$$\Delta y = y - t$$

$$\Delta a_2 = g'(a_2)\Delta y$$

$$\Delta z_1 = W_2 \Delta a_2$$

$$\Delta a_1 = f'(a_1)\Delta z_1$$

$$\nabla[W_1^\top, b_1] = \Delta a_1 [x^\top, 1]$$

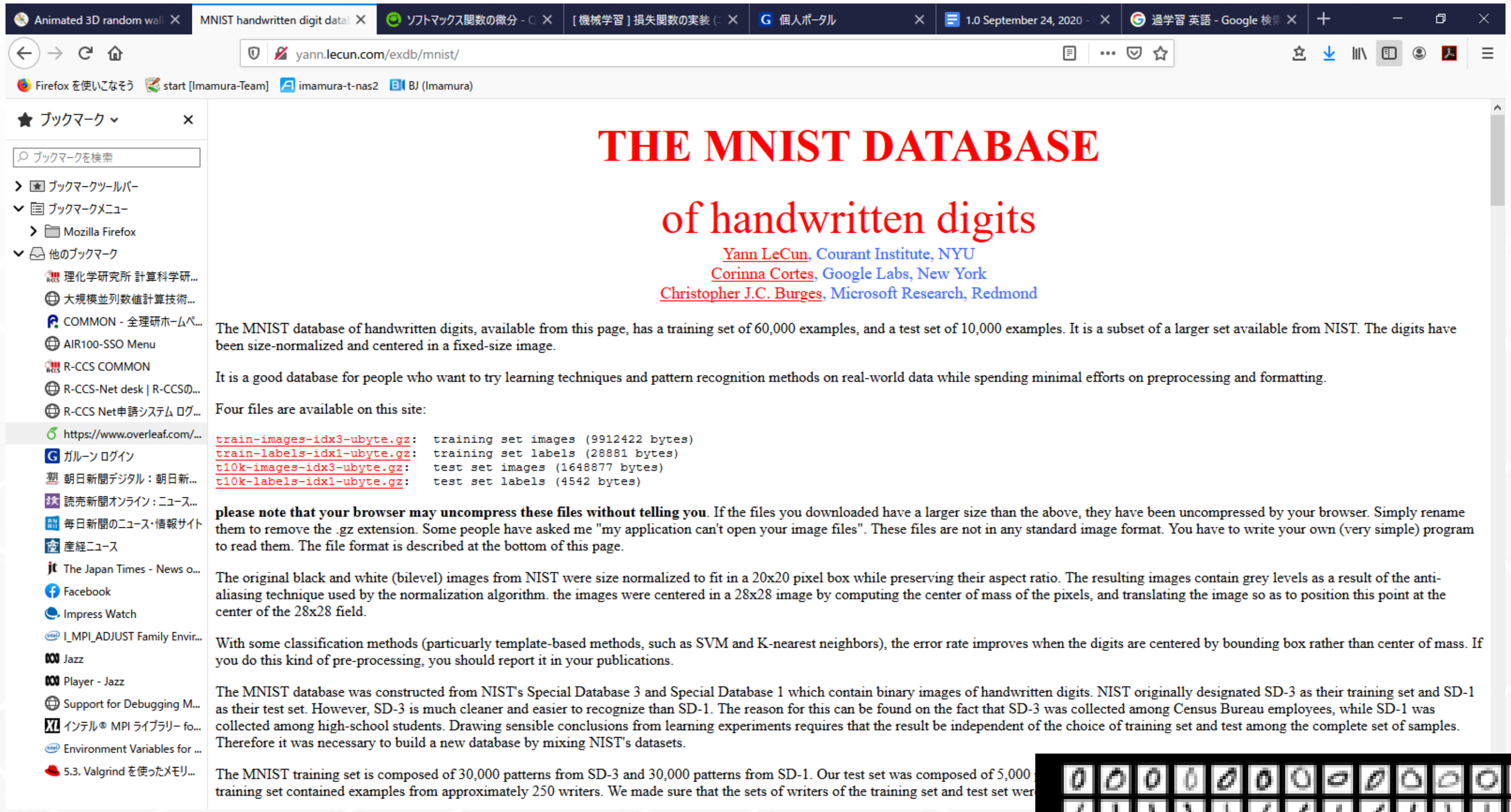
$$\nabla[W_2^\top, b_2] = \Delta a_2 [z_1^\top, 1]$$

Short break and demonstration

--MNIST handwritten letters and fashion--

The MNIST database

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



THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

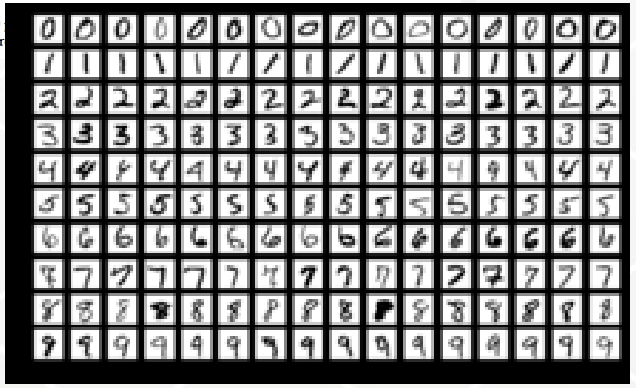
please note that your browser may uncompress these files without telling you. If the files you downloaded have a larger size than the above, they have been uncompressed by your browser. Simply rename them to remove the .gz extension. Some people have asked me "my application can't open your image files". These files are not in any standard image format. You have to write your own (very simple) program to read them. The file format is described at the bottom of this page.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

With some classification methods (particuarly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

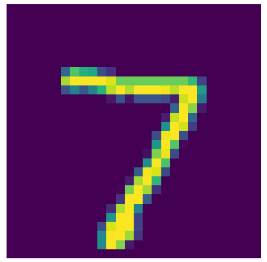
The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000
training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set wer

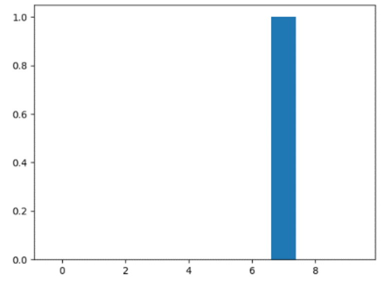


https://en.wikipedia.org/wiki/MNIST_database

Play a DNN toy on Google Colab!

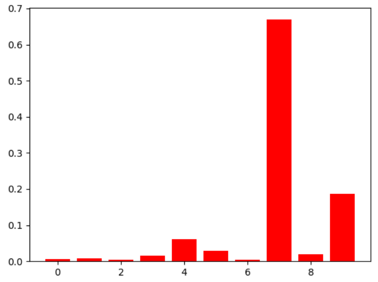
Overview of the structure of the Python script


Input figure



Input label

Neural Network



Output data

```
# forward
a1 = np.dot(x, W1) + b1
z1 = sigmoid(a1)
a2 = np.dot(z1, W2) + b2
y = softmax(a2)
```

```
# back
dy = (y - t) / batch_num
grads['W2'] = np.dot(z1.T, dy)
grads['b2'] = np.sum(dy, axis=0)
dz1 = np.dot(dy, W2.T)
da1 = sigmoid_grad(a1) * dz1
grads['W1'] = np.dot(x.T, da1)
grads['b1'] = np.sum(da1, axis=0)
```

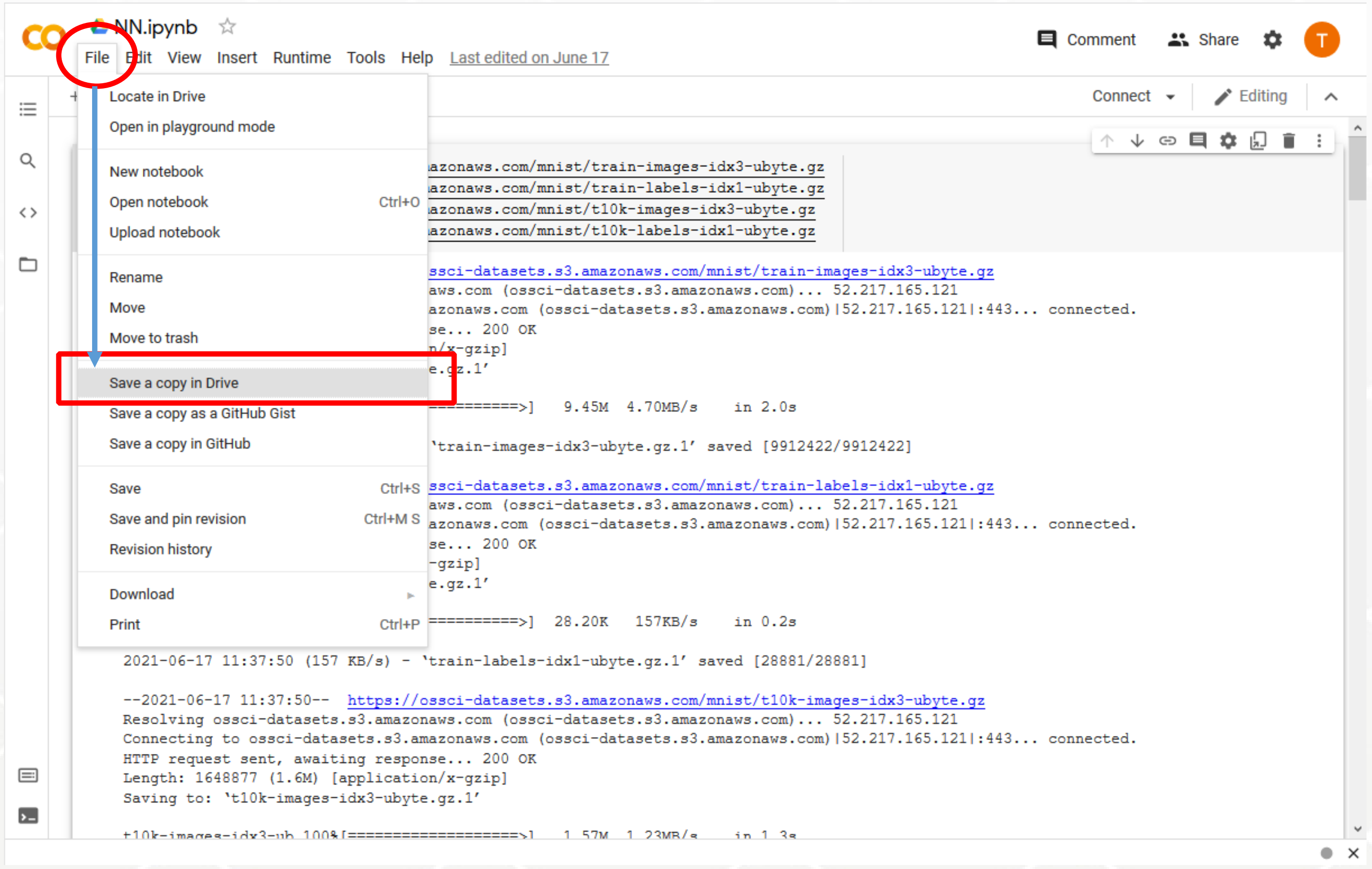
Python code of
forward and back propagation
(Look at **two_layer_net.py**)

```
NUM_DATAS = 60000 # The number of input data
NUM_TESTS = 10000 # The number of test data
max_iter = 10000 # The number of iteration
batch_size = 100 # The batch size mini batchlearning
lr=0.1 # learning ratio
```

Look at **NN.py**

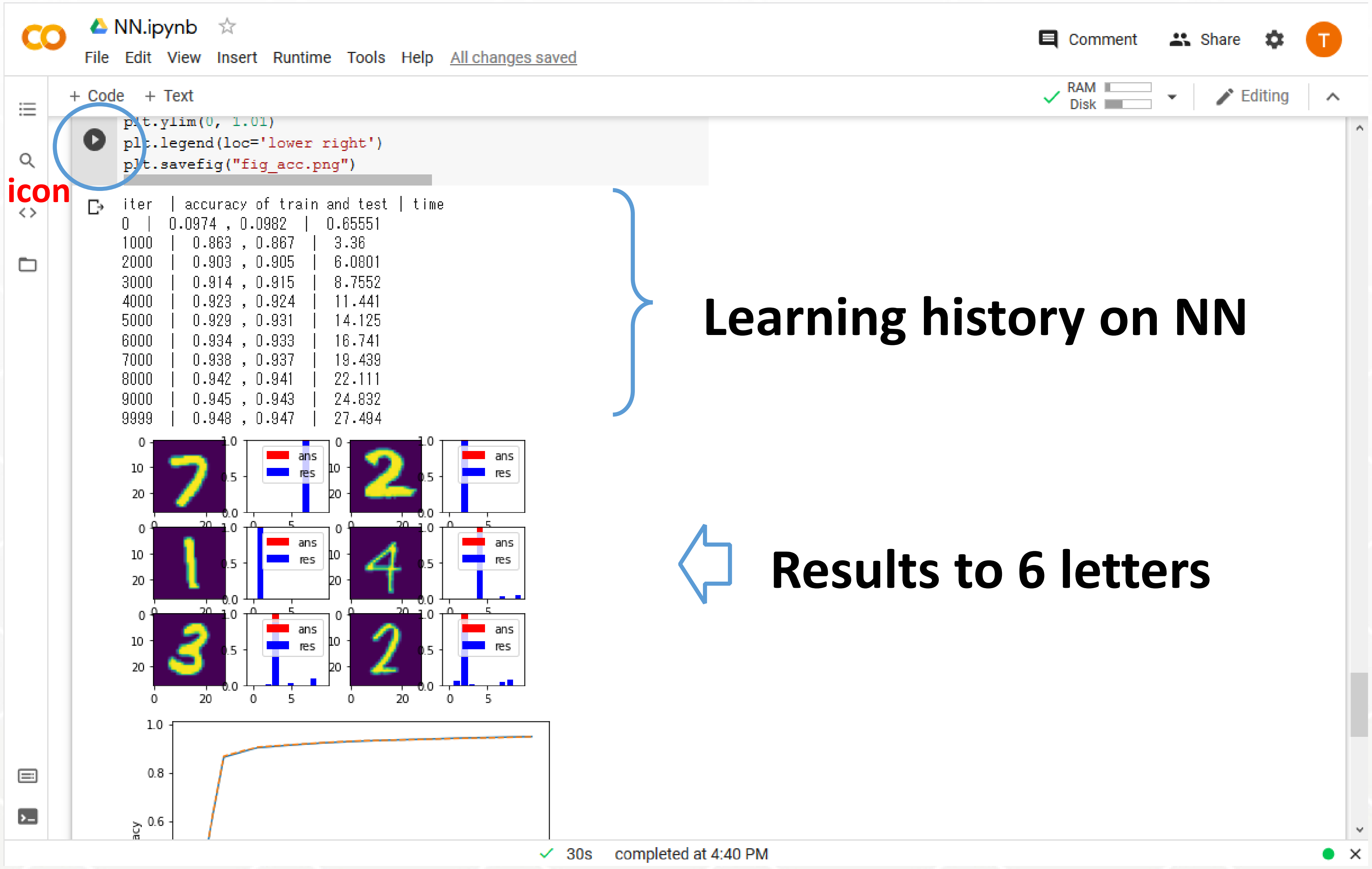
Link and copy

- **Click the link on Google Drive**
  - https://colab.research.google.com/drive/1sE42v9k0oJjweiowAJtRPb-h2R7h06K4?usp=sharing , then copy it on your drive, in order to make the code editable by yourself.



```
azonaws.com/mnist/train-images-idx3-ubyte.gz
azonaws.com/mnist/train-labels-idx1-ubyte.gz
azonaws.com/mnist/t10k-images-idx3-ubyte.gz
azonaws.com/mnist/t10k-labels-idx1-ubyte.gz

ssci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
aws.com (ossci-datasets.s3.amazonaws.com)... 52.217.165.121
azonaws.com (ossci-datasets.s3.amazonaws.com)|52.217.165.121|:443... connected.
se... 200 OK
n/x-gzip]
e.gz.1'

==========>]   9.45M  4.70MB/s    in 2.0s

'train-images-idx3-ubyte.gz.1' saved [9912422/9912422]

ssci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
aws.com (ossci-datasets.s3.amazonaws.com)... 52.217.165.121
azonaws.com (ossci-datasets.s3.amazonaws.com)|52.217.165.121|:443... connected.
se... 200 OK
-gzip]
e.gz.1'

==========>]  28.20K   157KB/s    in 0.2s

2021-06-17 11:37:50 (157 KB/s) - 'train-labels-idx1-ubyte.gz.1' saved [28881/28881]

--2021-06-17 11:37:50--  https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Resolving ossci-datasets.s3.amazonaws.com (ossci-datasets.s3.amazonaws.com)... 52.217.165.121
Connecting to ossci-datasets.s3.amazonaws.com (ossci-datasets.s3.amazonaws.com)|52.217.165.121|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1648877 (1.6M) [application/x-gzip]
Saving to: 't10k-images-idx3-ubyte.gz.1'
```

Run = push the “play” icon



```
plt.ylim(0, 1.01)
plt.legend(loc='lower right')
plt.savefig("fig_acc.png")
```

```
iter  | accuracy of train and test | time
0   |  0.0974 , 0.0982  |  0.65551
1000  |  0.863 , 0.867  |  3.36
2000  |  0.903 , 0.905  |  6.0801
3000  |  0.914 , 0.915  |  8.7552
4000  |  0.923 , 0.924  |  11.441
5000  |  0.929 , 0.931  |  14.125
6000  |  0.934 , 0.933  |  16.741
7000  |  0.938 , 0.937  |  19.439
8000  |  0.942 , 0.941  |  22.111
9000  |  0.945 , 0.943  |  24.832
9999  |  0.948 , 0.947  |  27.494
```

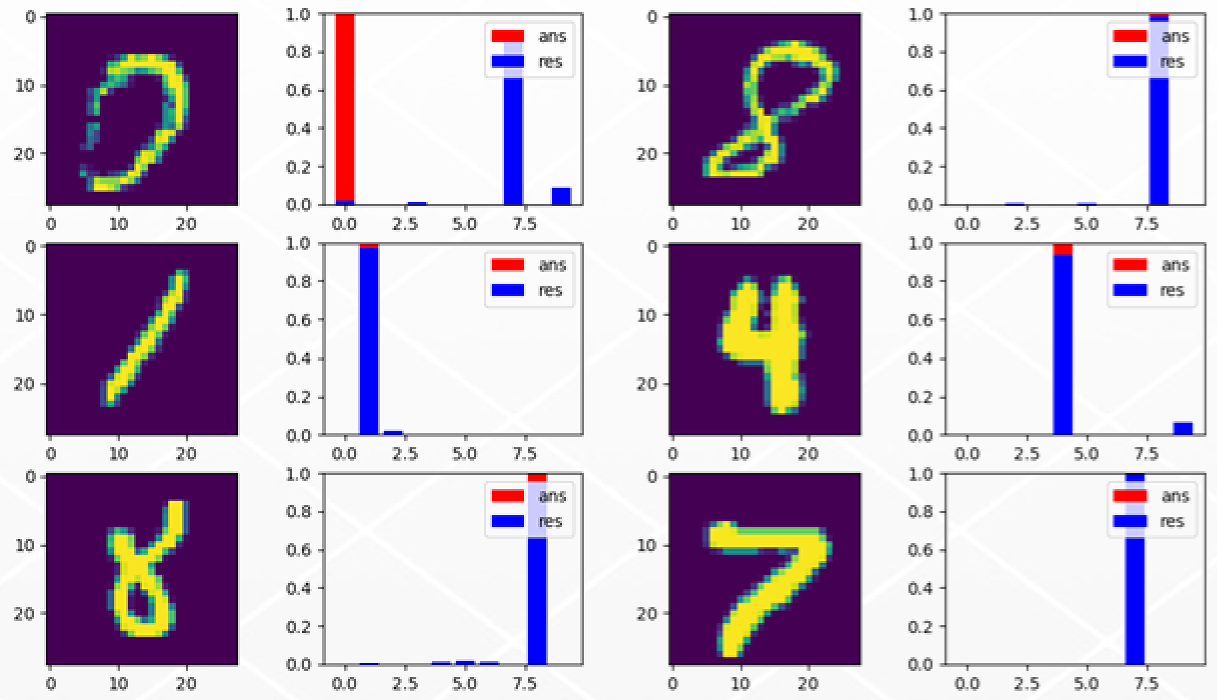
Learning history on NN

Results to 6 letters

How to read the example

Recognition fails
ans -> 0
max(res) -> 7

Success
ans = max(res)

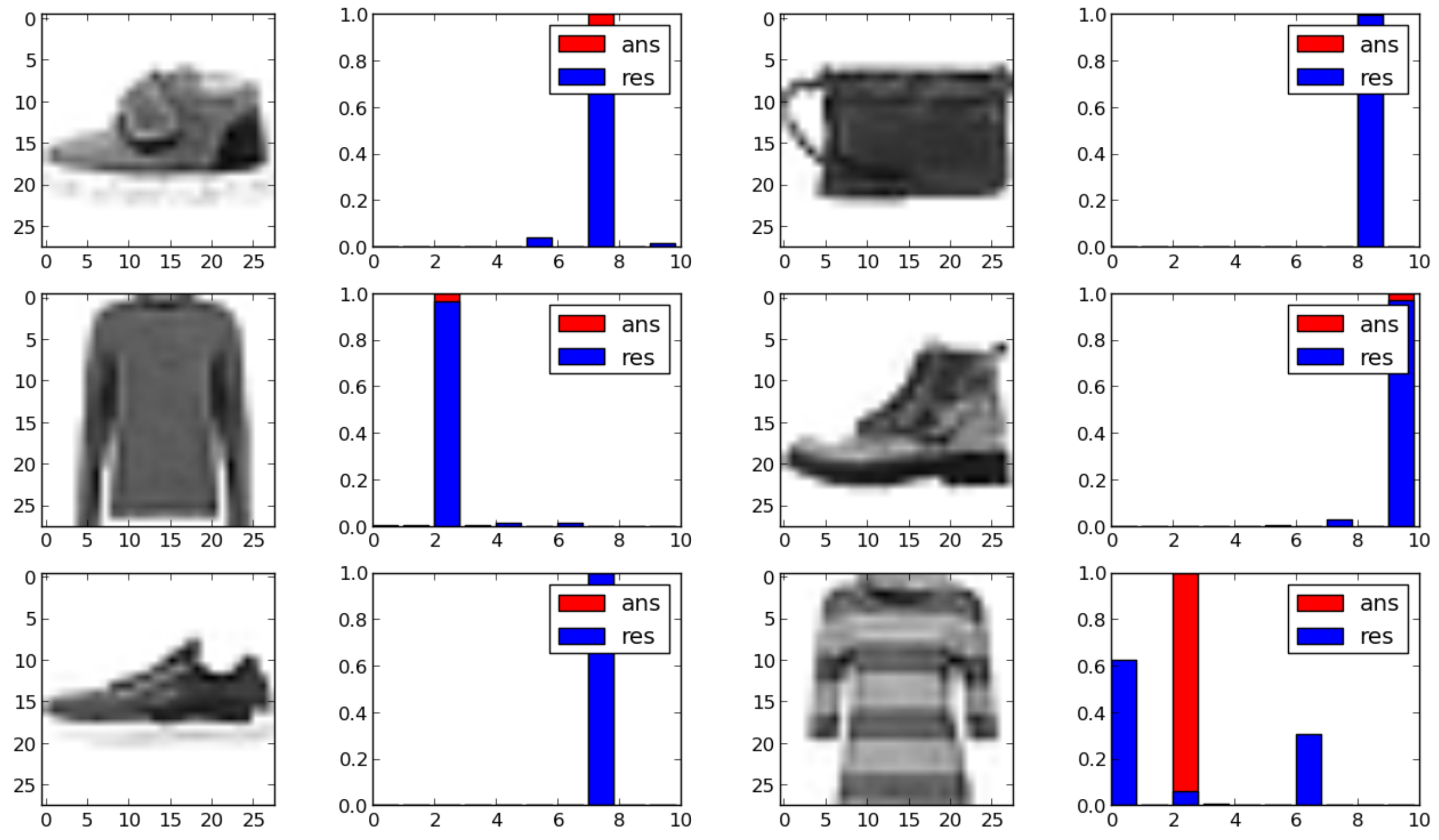


When you run the Python code on your PC. Do as follows:

$ python NN.py

https://www.nist.gov/itl/products-and-services/emnist-dataset
https://www.tensorflow.org/datasets/catalog/emnist

Another case (fashion MNIST)



https://github.com/zalandoresearch/fashion-mnist
https://www.tensorflow.org/tutorials/keras/classification

Exercise (30mins):

- **Edit load_mnist.py on the Google Colab editor as make available next lines and commented out other lines: “key_file =, … “ like,**

```
""""
key_file ={
    'x_train':'Fashion/train-images-idx3-ubyte.gz',
    't_train':'Fashion/train-labels-idx1-ubyte.gz',
    'x_test':'Fashion/t10k-images-idx3-ubyte.gz',
    't_test':'Fashion/t10k-labels-idx1-ubyte.gz’
}
“”””
```

- **More advanced, please access other xMNIST-data format**
  - KMNIST(kuzushiji) : http://codh.rois.ac.jp/kmnist/
  - Corrupted data : https://www.tensorflow.org/datasets/catalog/mnist_corrupted

Further Optimization of {w}

- **Stochastic / batched approach**
  - For every iteration, we update W by a full batch unit,

$$\Delta w_{ij} = -\epsilon \sum_{k \in \mathrm{dataset}} y_i^{(k)} y_j^{(k)} (1 - y_j^{(k)}) \frac{\partial E^{(k)}}{\partial y_j^{(k)}}$$

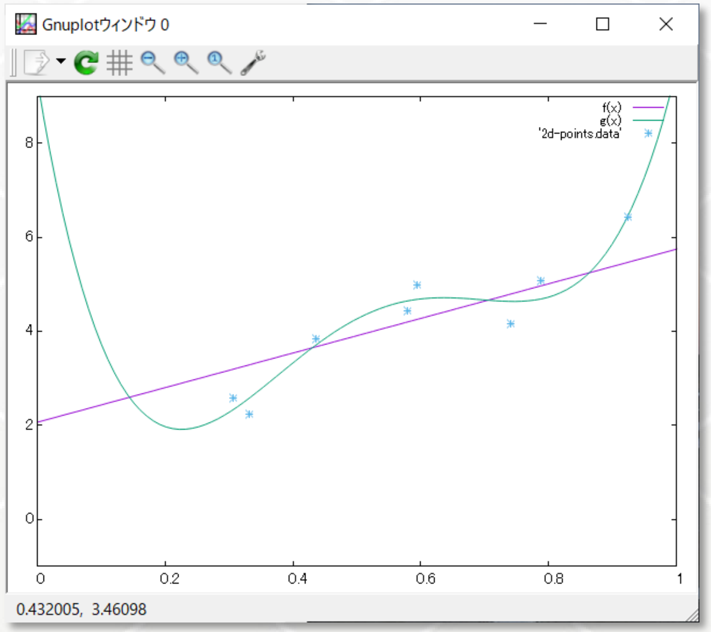
  - However, it might fail in local minimum or stagnation while GDM performs.

  - On stochastic SDM, at each iteration, error surface is estimated only with respect to a single example.
  - Major pitfall is a significant amount of iteration = time!
  - Another Mini-batch SD, which prevents such pros, divides data set as a bunch of batch sets.

$$\Delta w_{ij} = -\epsilon \sum_{k \in \mathrm{minibatch}} y_i^{(k)} y_j^{(k)} (1 - y_j^{(k)}) \frac{\partial E^{(k)}}{\partial y_j^{(k)}}$$

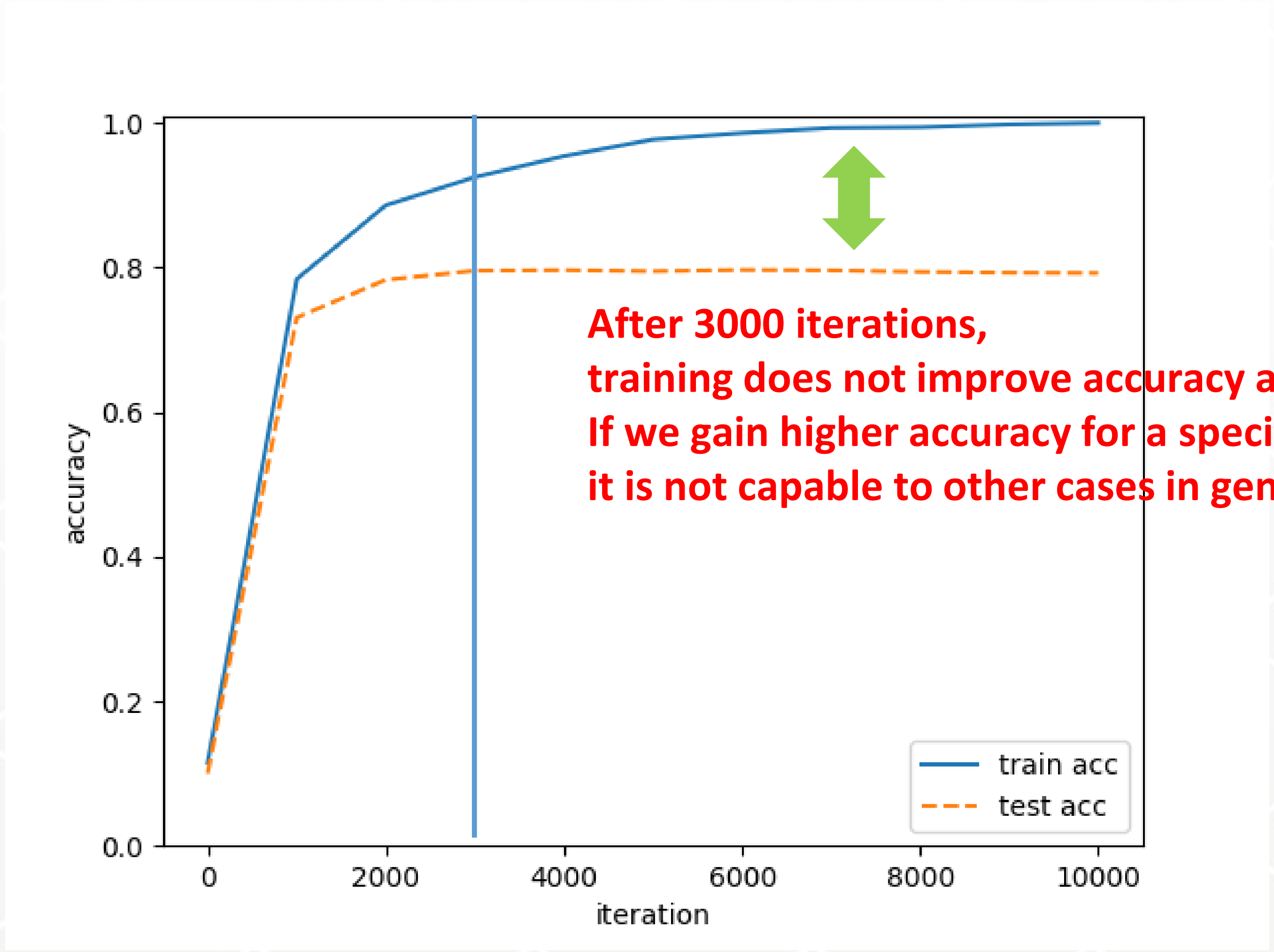
Training, Test, Validation, etc.

- **One problem:**
  - The model becomes very complicated.
  - What if we have a very complex model and a lot of training data sets, it is quite easy to fit the model.
  - However, new complex model does not generalize well.
- **Over-fitting**
  - Biggest challenge for the machine learning
  - *On large network, overfitting is commonplace.*



MNIST-fashion

- **Result of stagnated-learning history due to Overtraining.**



After 3000 iterations,
training does not improve accuracy anymore.
If we gain higher accuracy for a specific case,
it is not capable to other cases in general.

Another Error function in NN.py

- SSE: sum of squared error: explained
- Cross-Entropy

$$E = -\sum_k \left(t_k \log y_k + (1 - t_k)\log(1 - y_k)\right)$$

- In the case of classification problem, E should be simplified, t is one-hot vector and normalized |y|=1

$$E = -\sum_k t_k \log y_k \quad \Rightarrow \quad \frac{\partial E}{\partial y_k} = -\frac{1}{y_k}$$

- In NN.py, we take E and y=g(z) as cross-Entropy and softmax(), respectively.

$$\frac{\partial y_j}{\partial a_{1j}}\frac{\partial E}{\partial y_j} = (y_j(t_j - y_j))(-1/y_j) = y_j - t_j = (\Delta y)_j$$

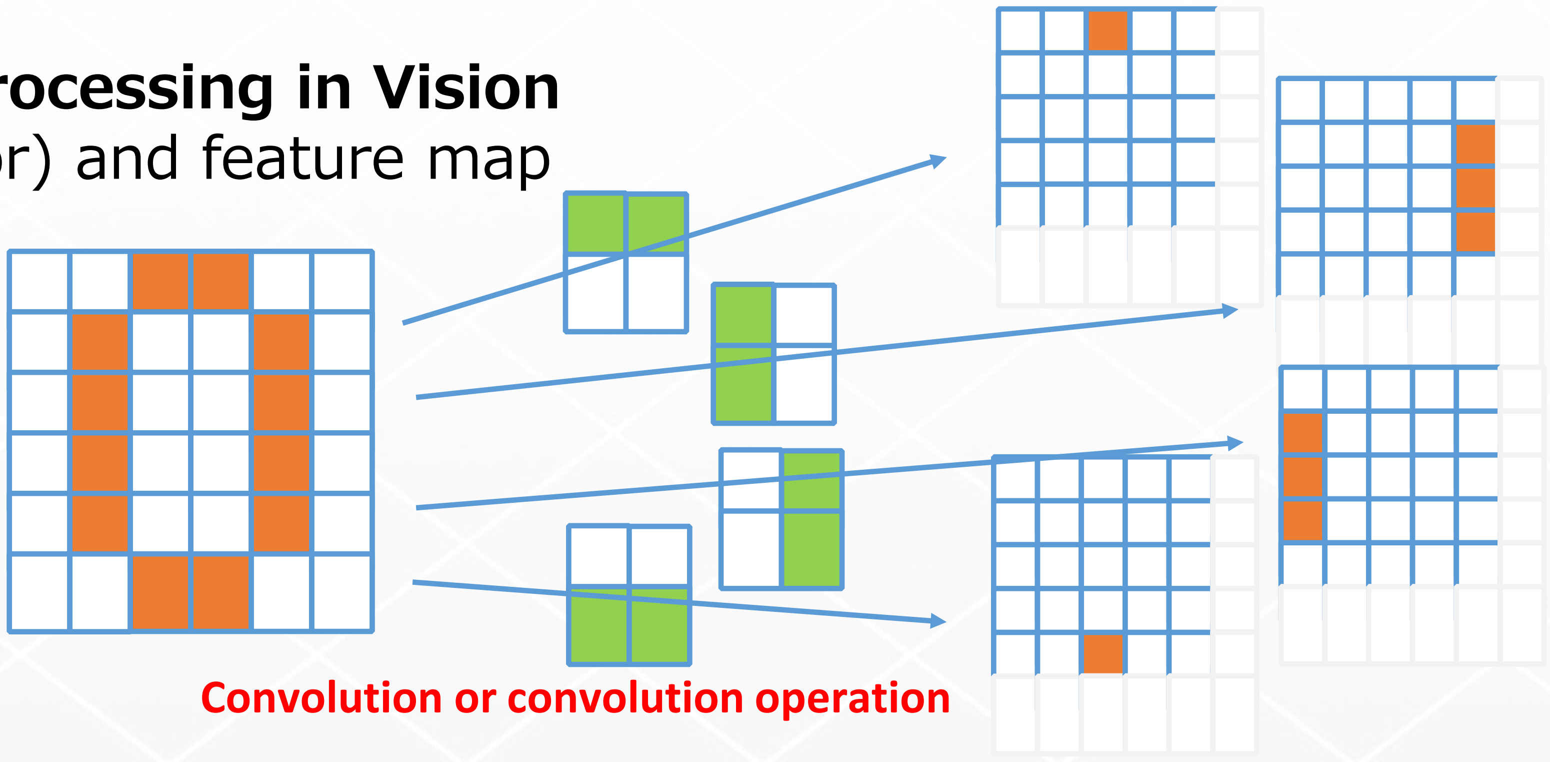
- Cross-entropy error shows often fast learning speed and better accuracy than SSE when classification.

For more advanced, image recognition case for practical uses

-- Just introductions

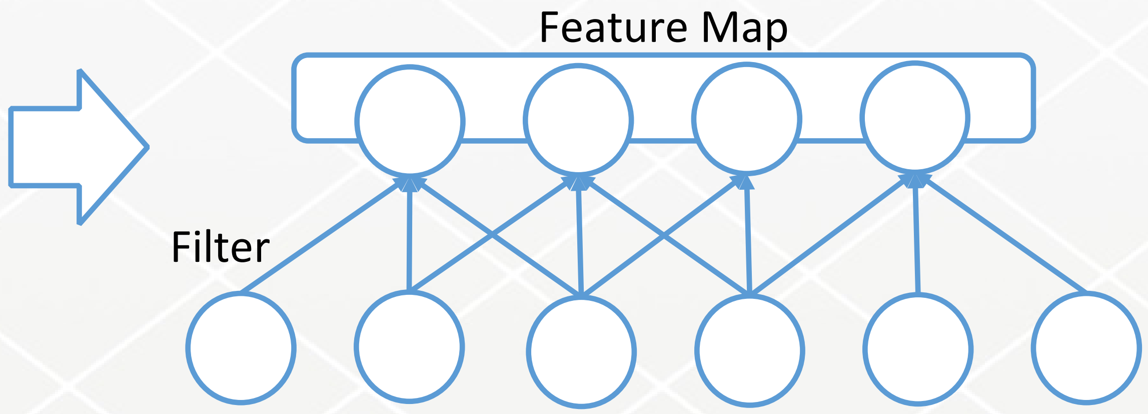
Convolution Neural Networks

- **Information processing in Vision**
  - Filter (detector) and feature map



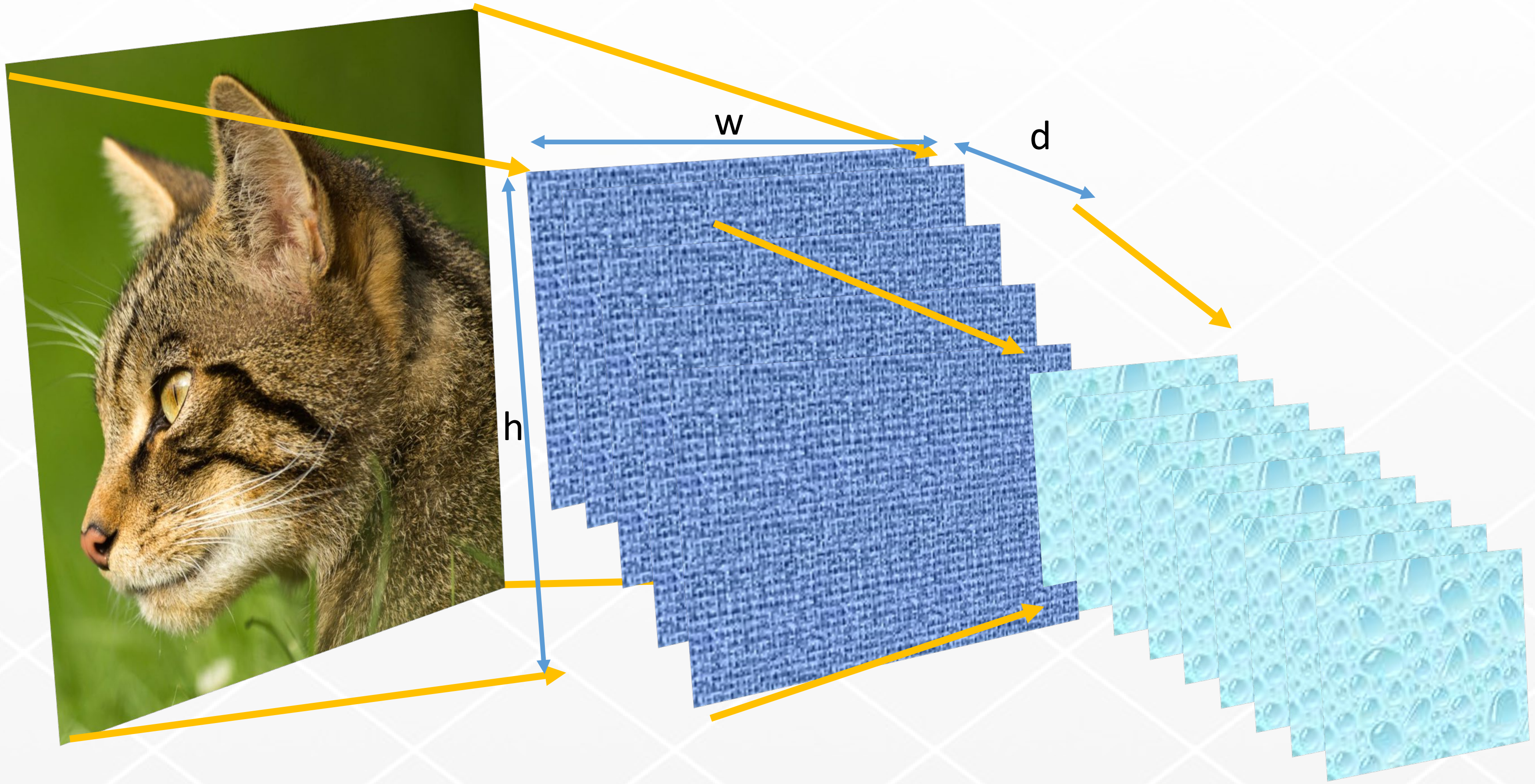
Convolution or convolution operation

$$m_{ij}^{k} = f\left(\underbrace{(W * x)_{ij}}^{\text{bitop \& sum}} + b^{k}\right)$$



Schematics

- (w,h,d,p): [width, height, depth, zero padding]



This network consists of a lot of parameters, part of them are called hyper-parameters.
These are also updated by similar techniques as back-propagation and descend sweep.
More flexible and user-friendly packages should be recommended for AI users.

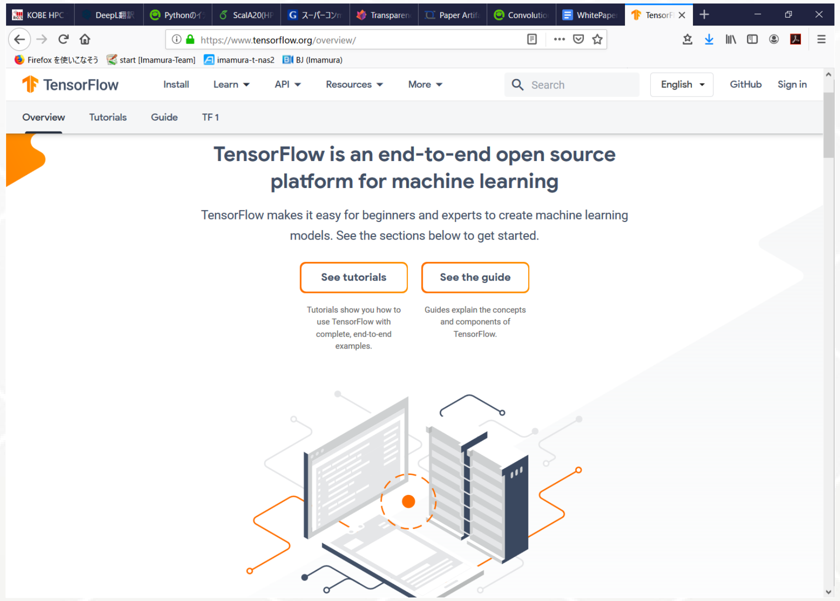
Appendix, More about DL software

- **TensorFlow (AI-engine) and Keras (high-level APIs)**
  - http://www.tensorflow.org/

  Building a deep and large deep learning model from scratch ⋯, one of the best or primary tool sets is TensorFlow by Google, 2015.

  - Open Source Software
  - Rich in tutorials
  - Many examples
  - A lot of available platforms

  - Accelerated by specific HWs



TensorFlow is an end-to-end open source platform for machine learning

TensorFlow makes it easy for beginners and experts to create machine learning models. See the sections below to get started.

Tutorials show you how to use TensorFlow with complete, end-to-end examples.

Guides explain the concepts and components of TensorFlow.