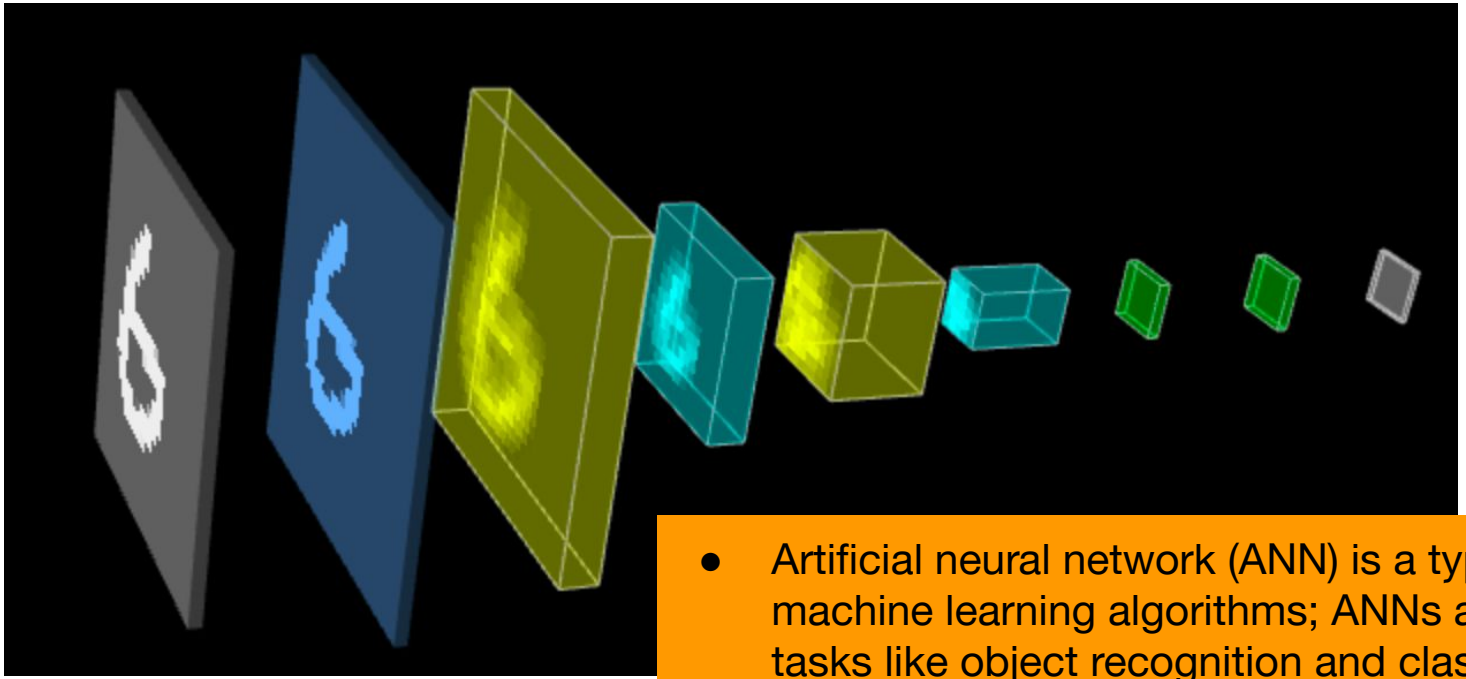
ICDS Spring 2025

Data Science: Part 3

Introduction to Artificial Neural Networks

An Online LeNet (click to take a try)



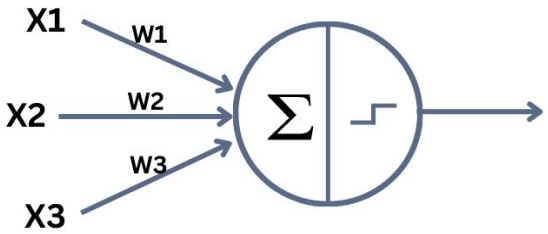
- Artificial neural network (ANN) is a type of supervised machine learning algorithms; ANNs are widely used in tasks like object recognition and classification.
- We will see the logic behind the ANN models and the code framework for building an ANN.

Agenda

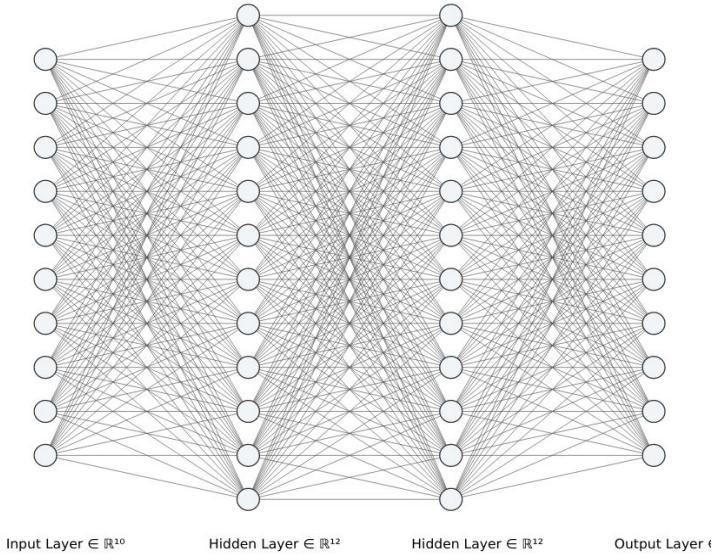
- Artificial neural network
  - Perceptron and multilayer networks
- Understanding the neural network
  - from a biological perspective
  - from a mathematical perspective
- Deep learning
  - Convolutional neural network
  - Applications: style transfer (in appendix)

A neural network is a “machine” built on perceptrons

- A perceptron is a simulation of a neuron, which is the primitive component of an artificial neural network.
- A network is a combination of perceptrons. (similar to the way that we build a computer with logic gates! The nature has a subtle consistency:)

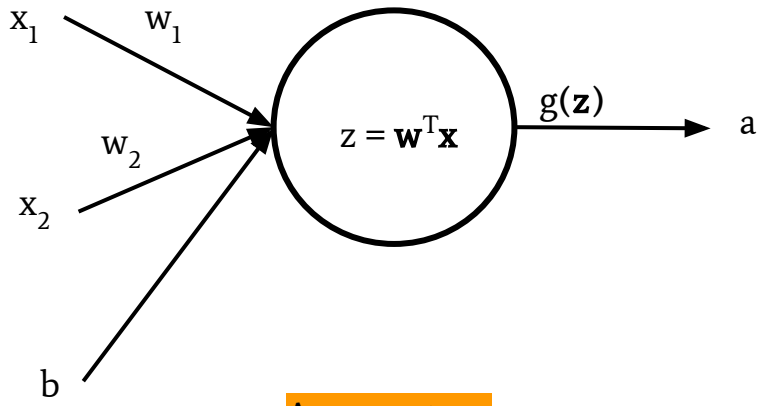


Single-layer perceptron



Multi-layer perceptron

Terminology



A perceptron

$z = \mathbf{w}^T\mathbf{x} + \mathbf{b} = w_1x_1 + w_2x_2 + b$

$a = g(z)$

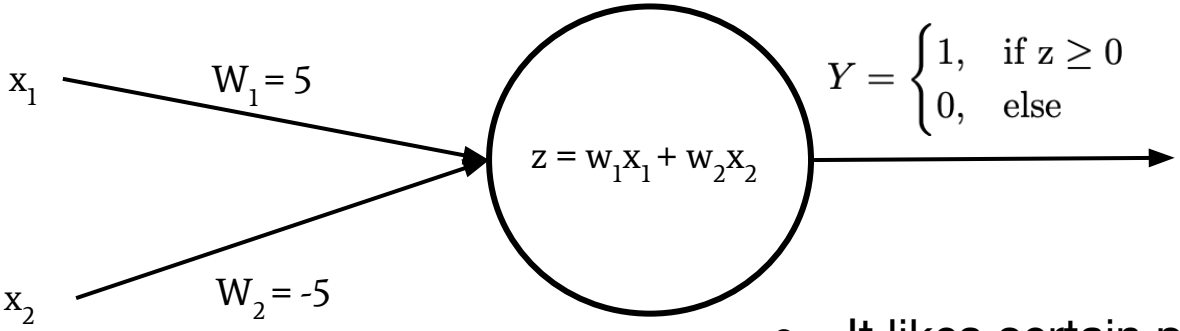
a: output (of a neuron)

g: activation function

w: input weights

b: bias (often used in practice)

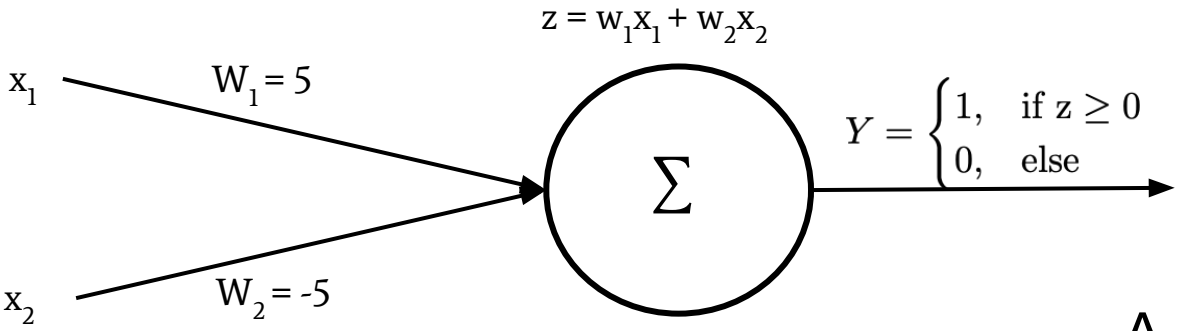
What does this perceptron do?



- It likes certain pattern, but hates the other

x_1	x_2	z	Y
1	-1	?	?
-1	1	?	?

What does this perceptron do?

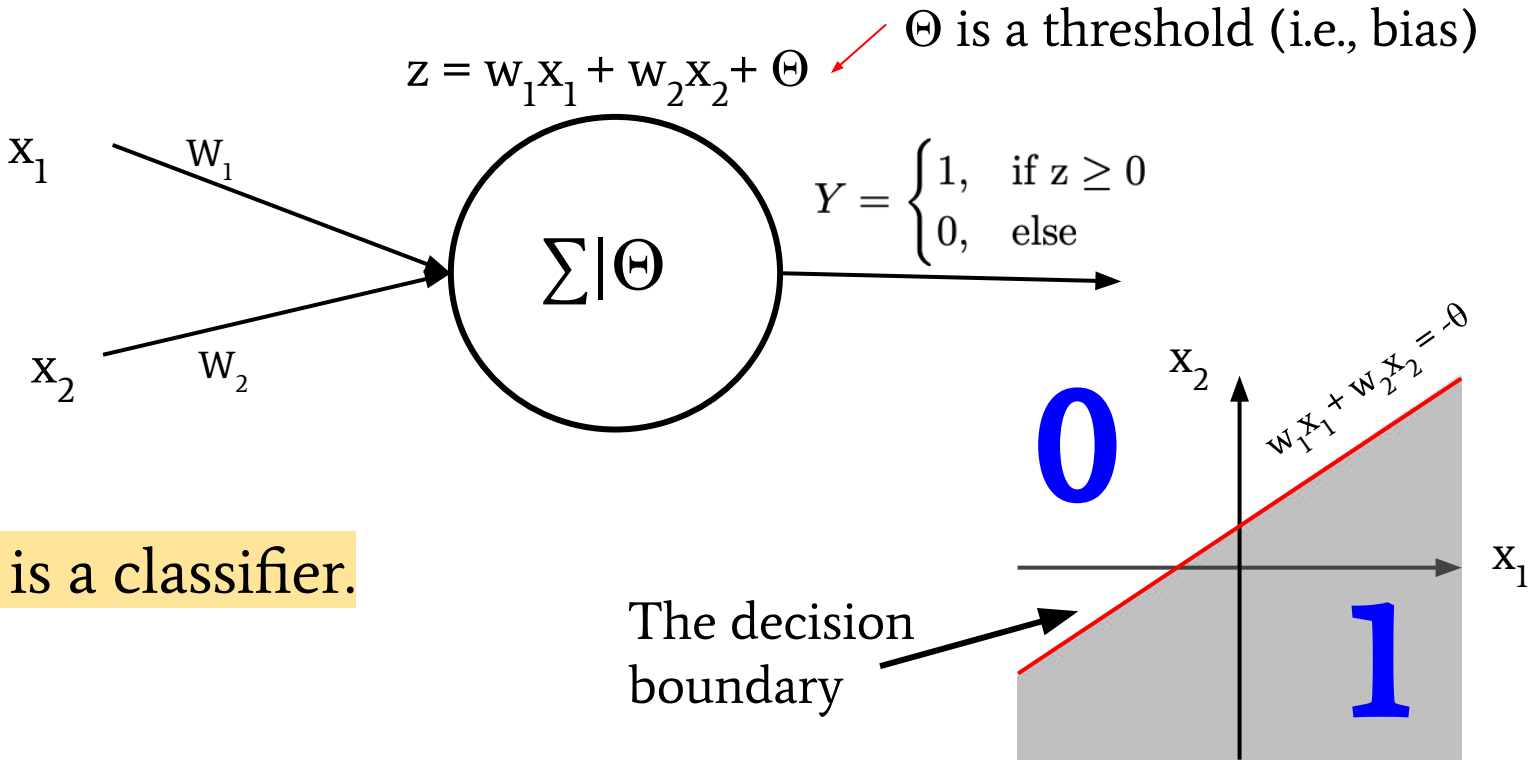


x_1	x_2	z	Y
1	-1	10	1
-1	1	-10	0

A neuron is a detector.

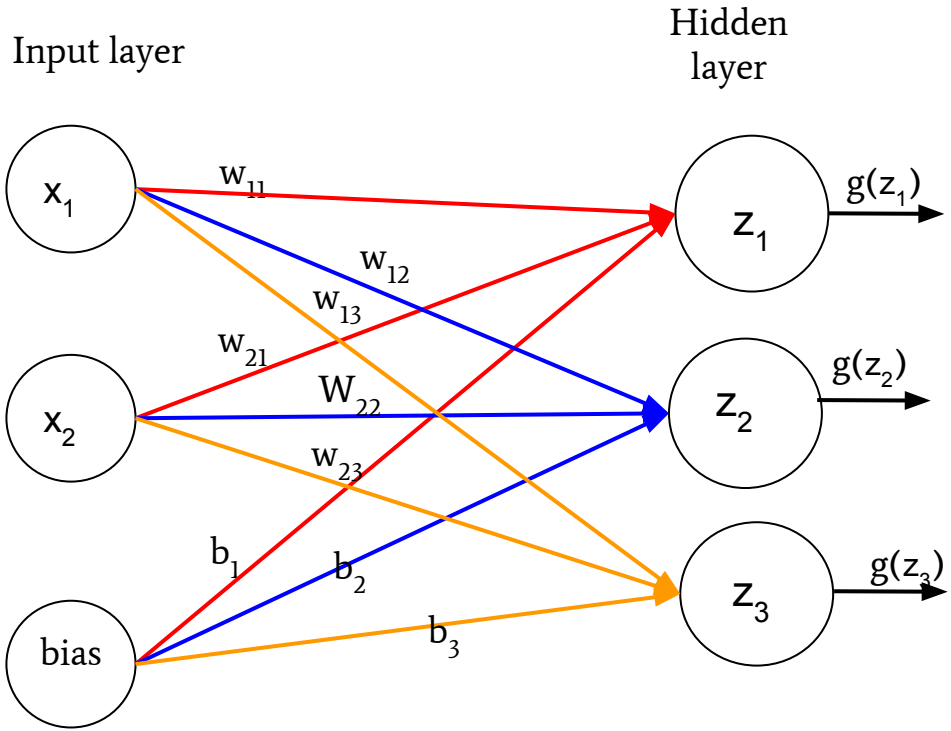
- Activated by certain pattern

What does this perception do?



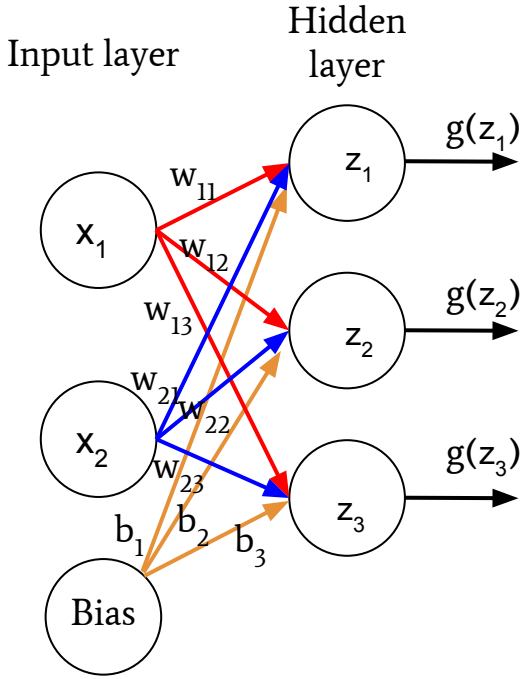
It is a classifier.

Forming a network



- A node in the hidden layer is a detector
- one detector → one feature
- n detectors → n feature → a feature vector $[g(z_1), g(z_2), ...]$

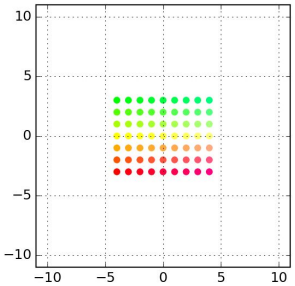
Z: a linear transformation + a shift



The input of the activation function g:

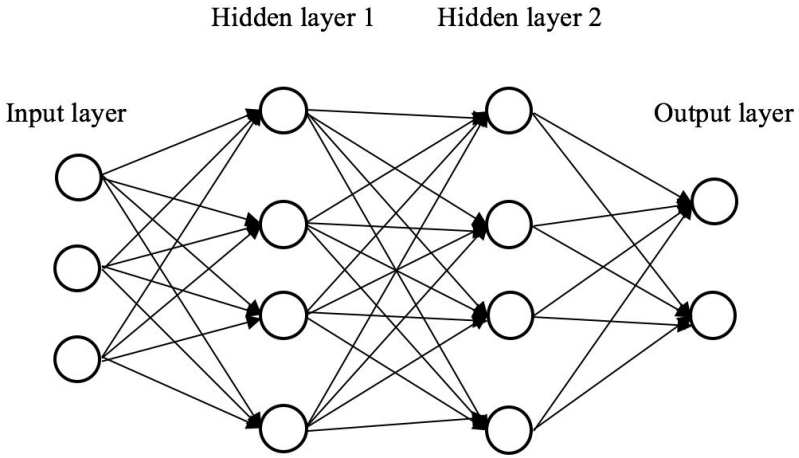
$$\mathbf{z} = W^T_{(2\times 3)}\mathbf{x} + \mathbf{b}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$



It performs a linear transformation by w and a translation by b.

Adding more layers



More hidden layers → more linear transformation on input data points.

- What does these layers bring out?

$$\text{layer 1: } \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1 \\ w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2 \\ w_{13}^{(1)} x_1 + w_{23}^{(1)} x_2 + b_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

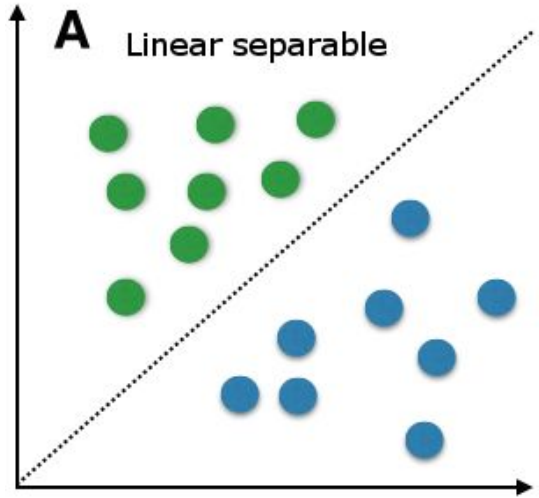
If the activation function is linear (e.g., g(z) = z), then, the network is still a linear transformation of the inputs. So, it cannot handle non-linear problem

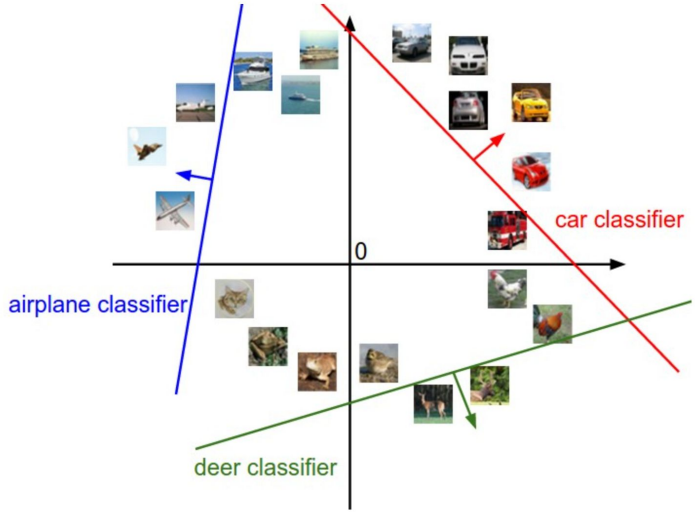
$$\text{layer 2: } \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \\ w_{13}^{(2)} & w_{23}^{(2)} & w_{33}^{(2)} \end{bmatrix} \times \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$$= \begin{bmatrix} (w_{11}^{(2)} w_{11}^{(1)} + w_{21}^{(2)} w_{12}^{(1)} + w_{31}^{(2)} w_{13}^{(1)}) x_1 + (w_{11}^{(2)} w_{21}^{(1)} + w_{21}^{(2)} w_{22}^{(1)} + w_{31}^{(2)} w_{23}^{(1)}) x_2 + (w_{11}^{(2)} + w_{21}^{(2)} + w_{31}^{(2)}) b_1 \\ (w_{12}^{(2)} w_{11}^{(1)} + w_{22}^{(2)} w_{12}^{(1)} + w_{32}^{(2)} w_{13}^{(1)}) x_1 + (w_{12}^{(2)} w_{21}^{(1)} + w_{22}^{(2)} w_{22}^{(1)} + w_{32}^{(2)} w_{23}^{(1)}) x_2 + (w_{12}^{(2)} + w_{22}^{(2)} + w_{32}^{(2)}) b_2 \\ (w_{13}^{(2)} w_{11}^{(1)} + w_{23}^{(2)} w_{12}^{(1)} + w_{33}^{(2)} w_{13}^{(1)}) x_1 + (w_{13}^{(2)} w_{21}^{(1)} + w_{23}^{(2)} w_{22}^{(1)} + w_{33}^{(2)} w_{23}^{(1)}) x_2 + (w_{13}^{(2)} + w_{23}^{(2)} + w_{33}^{(2)}) b_3 \end{bmatrix}$$

Limits of linear classifiers

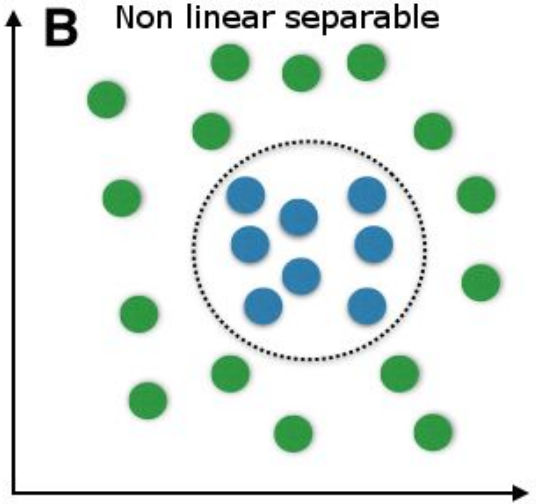
- Change the weights will change the angle of the line
- Change the bias will move the line right or left
- Linear classifiers can only handle linear separable problems

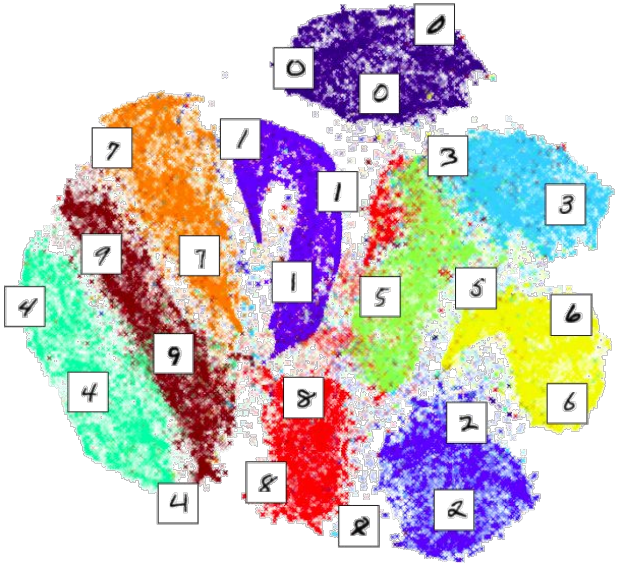




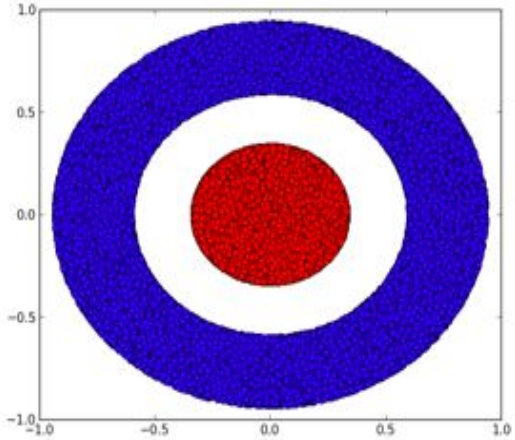
Nonlinear separable problems

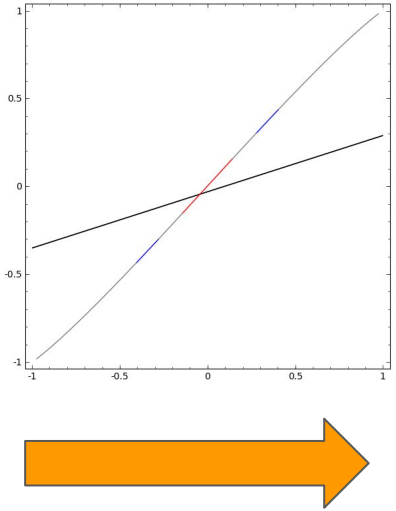
- Linear classifiers cannot solve them directly ⇒ But we can convert them to linear separable problems by using nonlinear activation functions

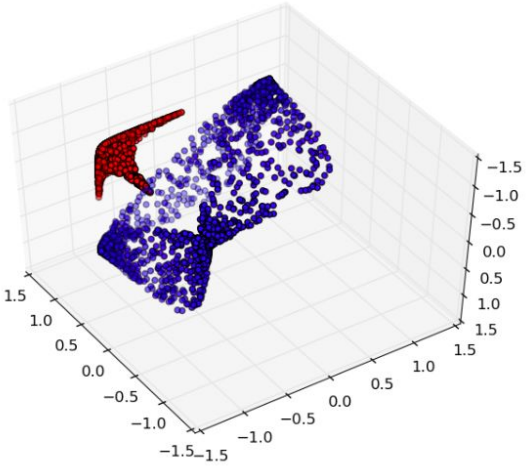




Adding non-linearity



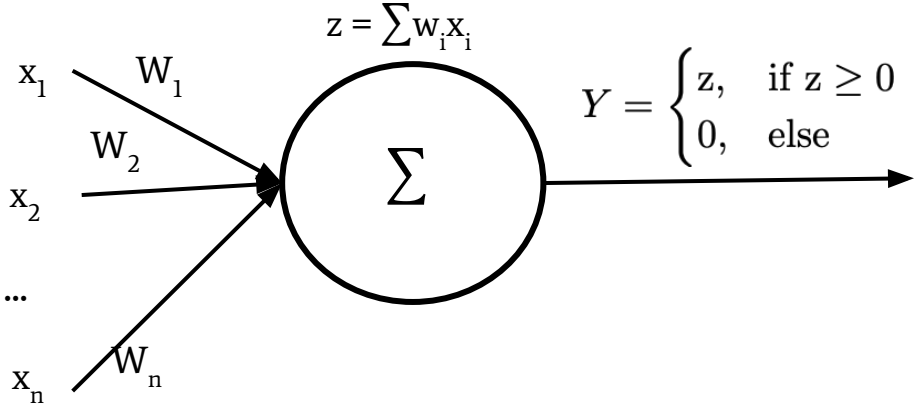


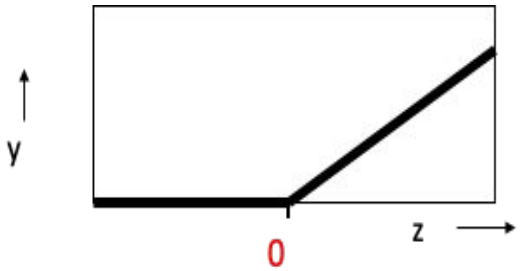


- Adding layers → stretch, rotate and shift
- nonlinear activation functions → folds, break, cut ...

Nonlinear activation functions

- Rectified Linear Unit (ReLU)

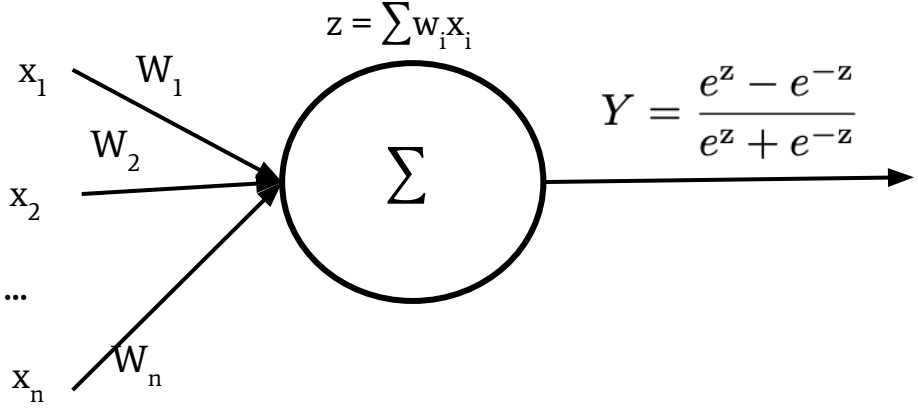


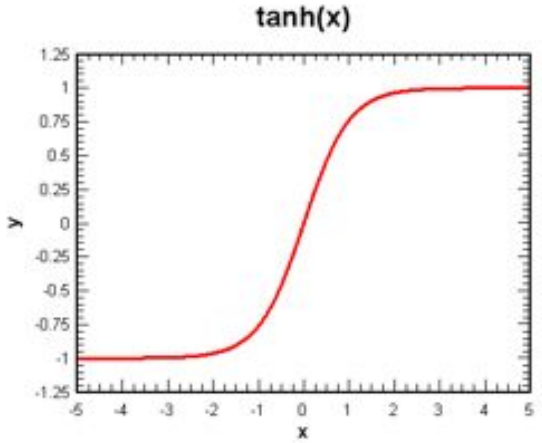


Rectified linear neuron

Nonlinear activation functions

- Tanh (hyperbolic tangent)

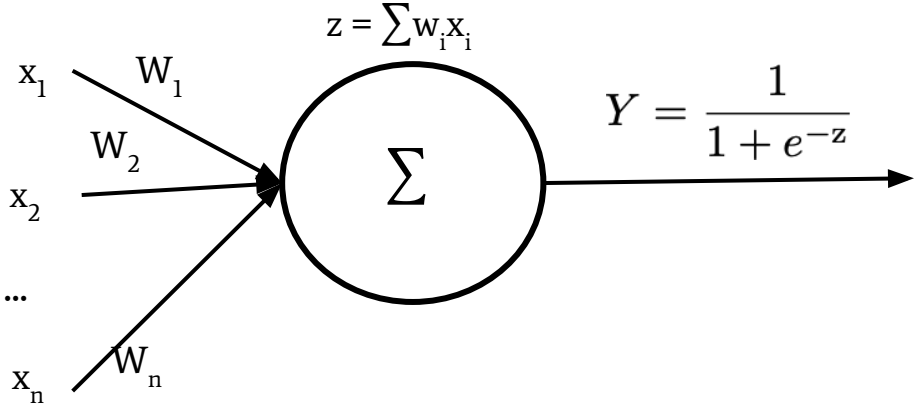


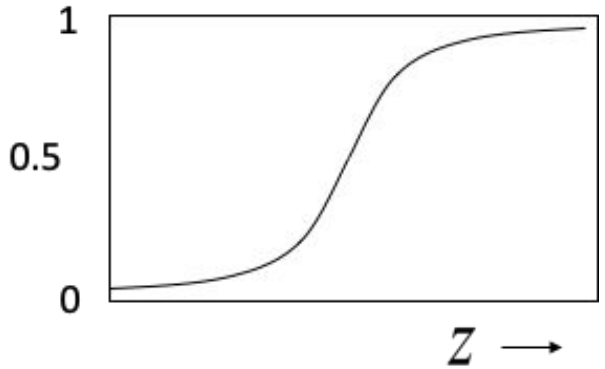


tanh neuron

Nonlinear activation functions

- Sigmoid



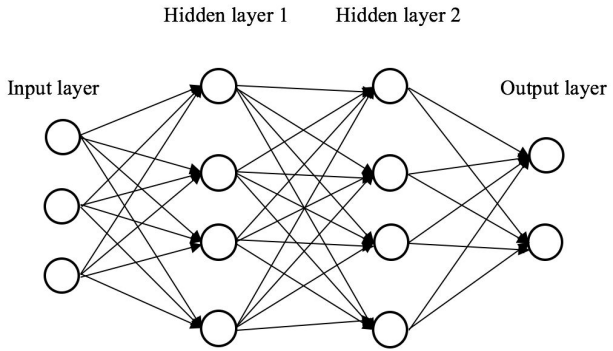


Sigmoid neuron

Your Turn

Scan to answer!

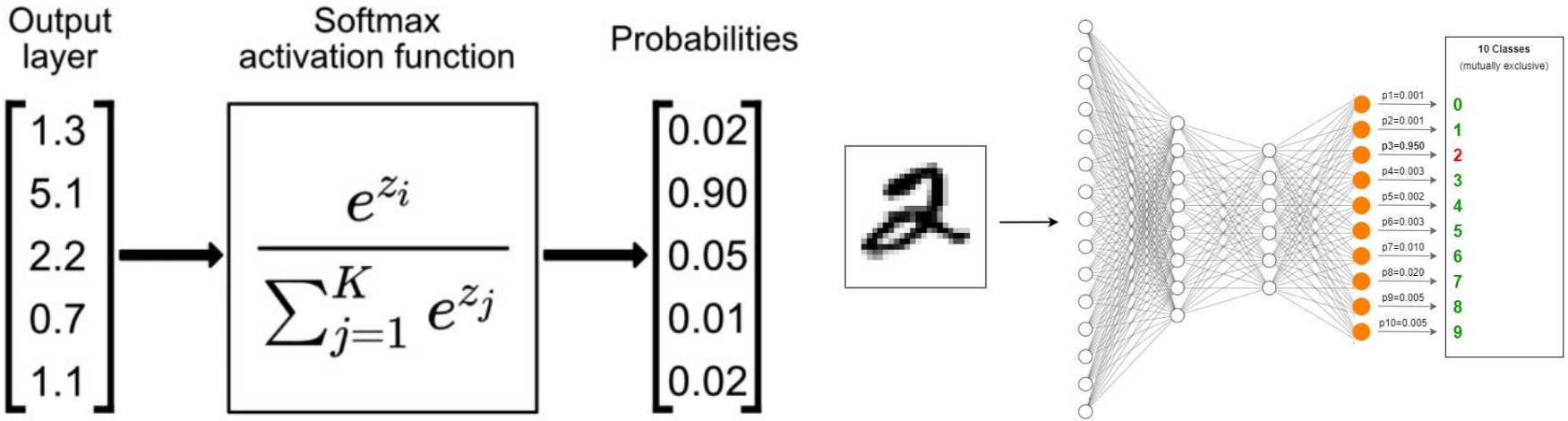
1. Which activation function should we use if we want to build a neural network for the commercial recommendation problem?
   a. ReLU
   b. Tanh
   c. Sigmoid
   d. Softmax



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0

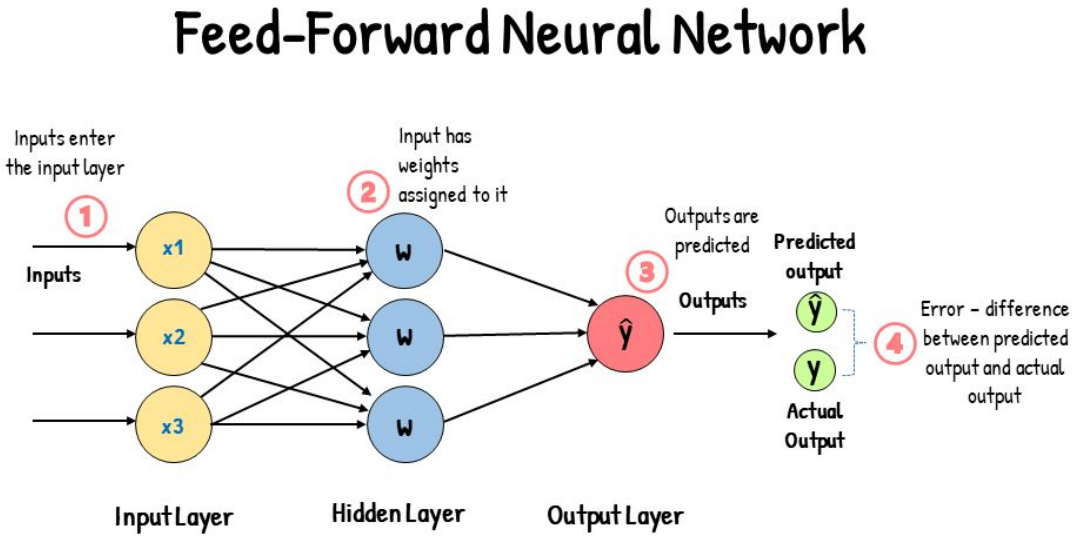
Nonlinear activation functions

- Softmax



Train a neural network, i.e., learn the W

Forward propagation: the inputs ⇒ go through the layers ⇒ output

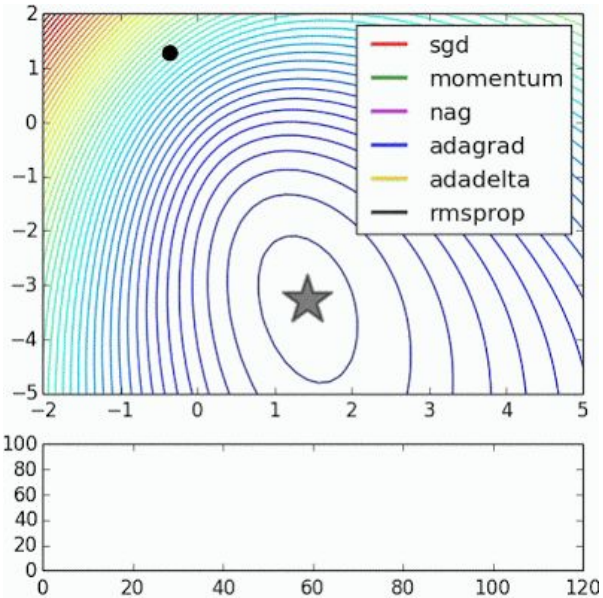


Learning W (i.e., training the neuron network)

The loss function: $J(\Theta) = \frac{1}{M}\sum_{i=1}^{M} \ell(y^{(i)}, f(\mathbf{x}^{(i)}))$, where $\Theta = \{\mathbf{w}, b\}$.

Minimizing J(Θ) → gradient descent.

- Starting from some randomly chosen Θ
- Increasing by $\Theta = \Theta - \lambda \frac{\partial J}{\partial \Theta}$

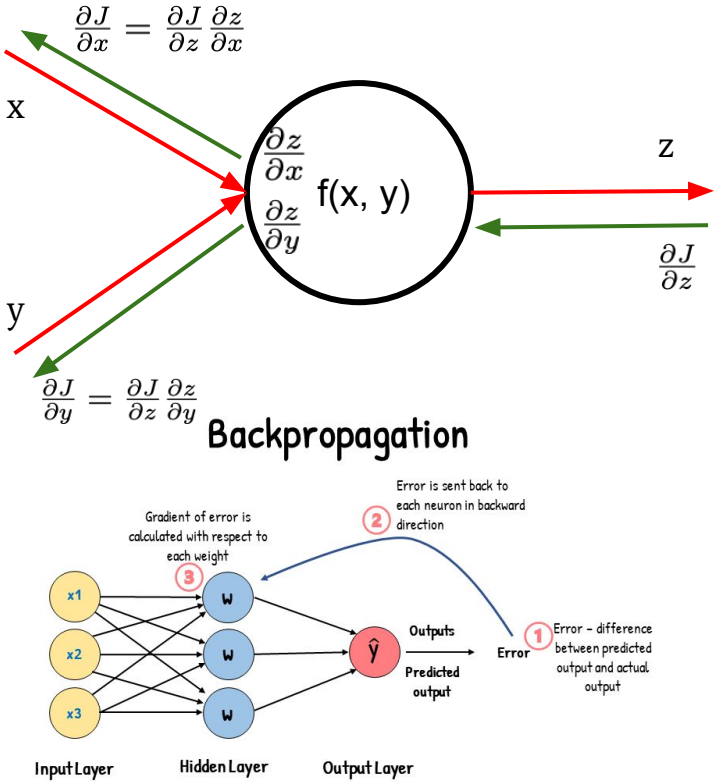


Different ways for adapting the learning rate

Updating W: Backpropagation

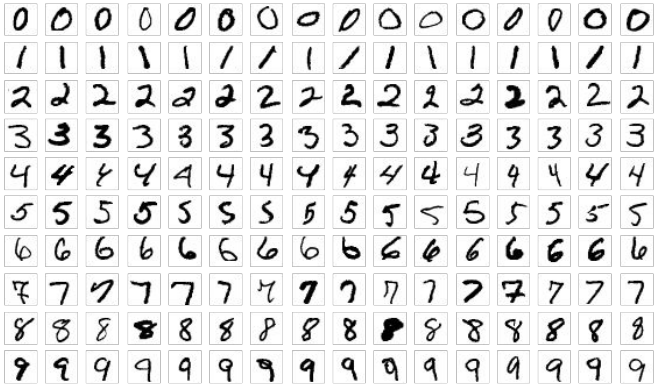
For each neuron, there are some inputs and an output (denote as x, y and z in the following example).

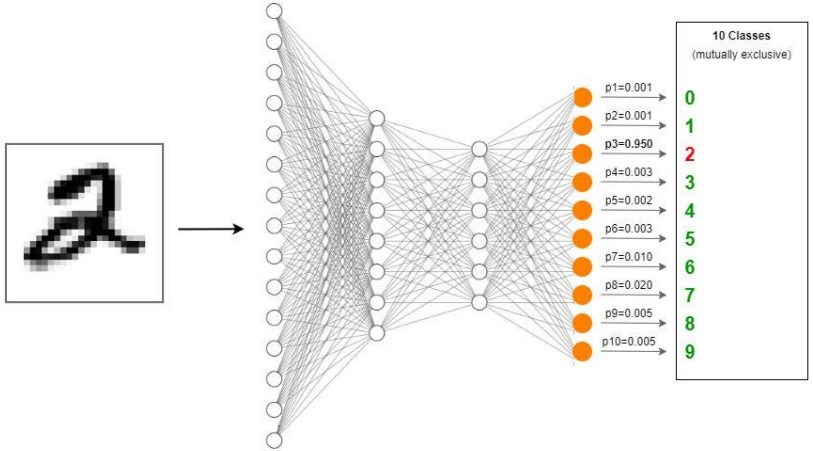
- The network propagates the signal of the input data forward to the output layer (following red arrows), then,
- It back propagates information about the error, in reverse through the network (following green arrows)
- “W”s are altered during backpropagation ⇒ doing gradient descent for each w layers by layers



Building a MLP for the MNIST dataset

- MLP: Multi-Layer Perception
- MNIST is a dataset of handwritten digits, often used in demos (proposed by Yann LeCun et al)





Install Pytorch using conda

Open a terminal / CMD, input the followings code in yellow one after another

1. `conda create --name pytorch` (or any name for the environment you prefer) # to create a new environment for Pytorch
2. `activate pytorch` (or the name of your environment) # activate the environment you create
3. #Install matplotlib and numpy (see the next slide)
4. `conda install pytorch::pytorch torchvision torchaudio -c pytorch` # install pytorch to the environment

For more conda shell command, please refer to this link: CONDA CHEAT SHEET

Install Pytorch in conda env

```
(base) bing@Xianbins-MacBook-Pro ~ % conda create --name pytorch2
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 22.11.1
  latest version: 23.7.1

Please update conda by running

    $ conda update -n base -c conda-forge conda

Or to minimize the number of packages updated during conda update use

     conda install conda=23.7.1



## Package Plan ##

  environment location: /Users/bing/opt/anaconda3/envs/pytorch2


Proceed ([y]/n)? y
```

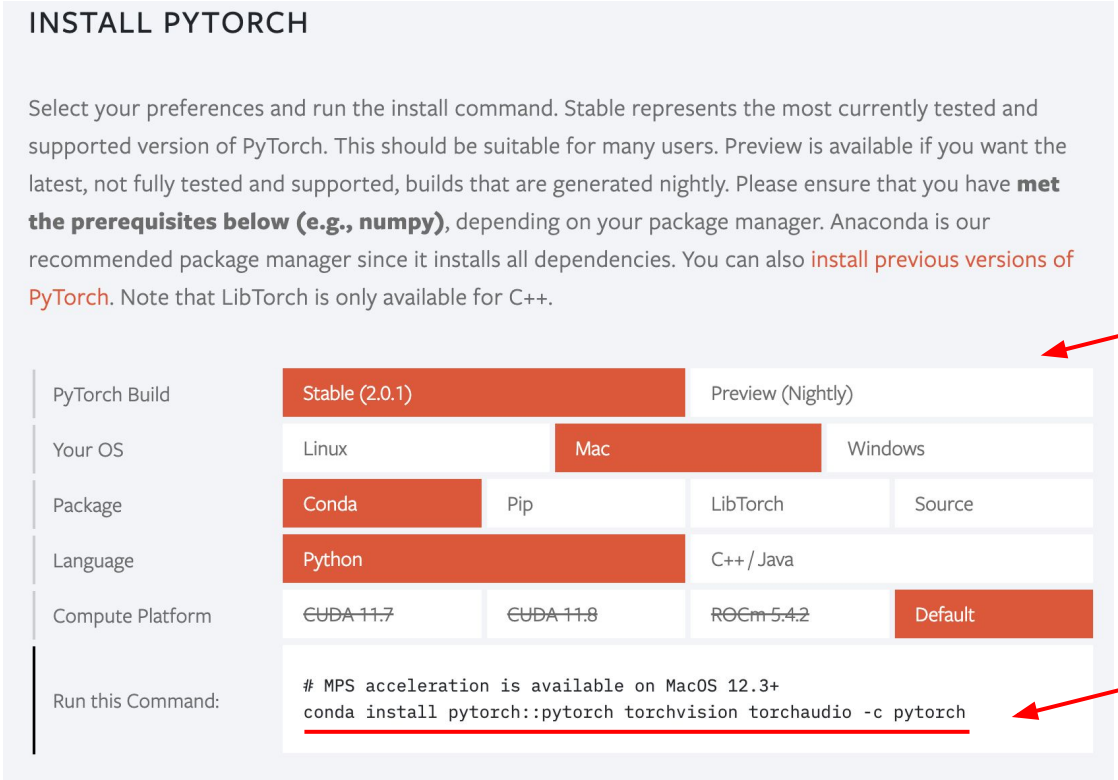
Press y to proceed

Create an environment named pytorch2 (you may use other names you like)

You need install matplotlib and numpy: (please copy and run the followings to the shell one after another)

- `conda install -c conda-forge matplotlib`
- `conda install -c anaconda numpy`

Install Pytorch: official page



A installation guide can be found at the official website: https://pytorch.org/

Choose the build and the settings of your platform

Copy and paste it to your terminal/cmd

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 50)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(50, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        return F.log_softmax(self.fc3(x), dim=1)
```

Set the connections of the network

- self.fc1: connection between 1st layer and 2nd layer: 784 ⇒ 50;
- self.fc1_drop: randomly choose 20% of the weights in fc1, ignore them (i.e., set them to 0); this can increase the generalizability of the network (i.e., improve the model's performance on unseen data).

Set the forward propagation of the network

- x: the input data
- F.relu: the Relu activation function
- F.log_softmax: softmax function to convert the output values of self.fc3 to probabilities (i.e., values in [0, 1])

Create an instance of the Net class;
.to(device): put it to RAM of CPU or RAM of GPU

```
model = Net().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
criterion = nn.CrossEntropyLoss()
```

Choose a loss function:

- Pytorch provides several loss functions; here, we use cross entropy loss function

Set the optimizer for the loss function:

- Here, we use SGD to update all model.parameters
- lr: the learning rate
- momentum: an argument of SGD algorithm

```python
def train(epoch, log_interval=200):
    # Set model to training mode
    model.train()
    # Loop over each batch from the training set
    for batch_idx, (data, target) in enumerate(train_loader):
        # Copy data to GPU if needed
        data = data.to(device)
        target = target.to(device)
        # Zero gradient buffers
        optimizer.zero_grad()
        # Pass data through the network
        output = model(data)
        # Calculate loss
        loss = criterion(output, target)
        # Backpropagate
        loss.backward()
        # Update weights
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data.item()))
```

Forward propagation (i.e., feed the network with data)

Compute the loss

Backpropagation (i.e., compute the partial derivatives) and updating the weights.

Agenda

- Artificial neural network
  - Perceptron and multilayer networks
- **Understanding the neural network**
  - from a biological perspective
  - from a mathematical perspective
- Deep learning
  - Convolutional neural network
  - Applications: style transfer (in appendix)

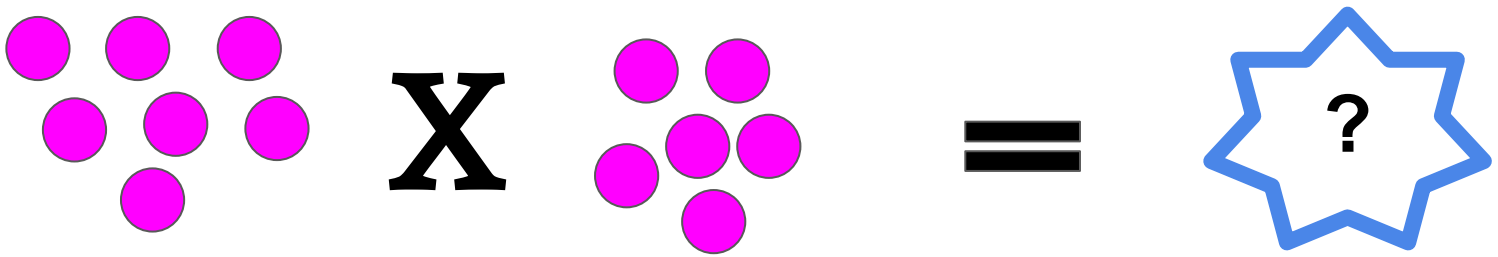
Neural network from a biological perspective

- Feature is all you need

What is it?

7 X 6 = 

What is it?



What is it?

VII X VI = 

What is it?

七 X 六 = 

From features to cognition: biological evidence

Brains recognize features; no matter they are from eyes or ears:

- Example from neuroscience: rewiring a ferret's auditory area to visual signals ⇒ The ferret can "see" by its "ears". (Von Melchner et al., 2000)
  - Visual signals and auditory signals might be recognized in the same way.
  - It is the feature (e.g., visual feature, or auditory feature) determines what it perceives.

Features (i.e., data representations) are critical

- Proper representation → quick answer
- Improper representation → confusion

In machine learning algorithms, things are the same.

- Good feature → good performance
- Unsuitable feature → meaningless outputs

How do animals extract features?

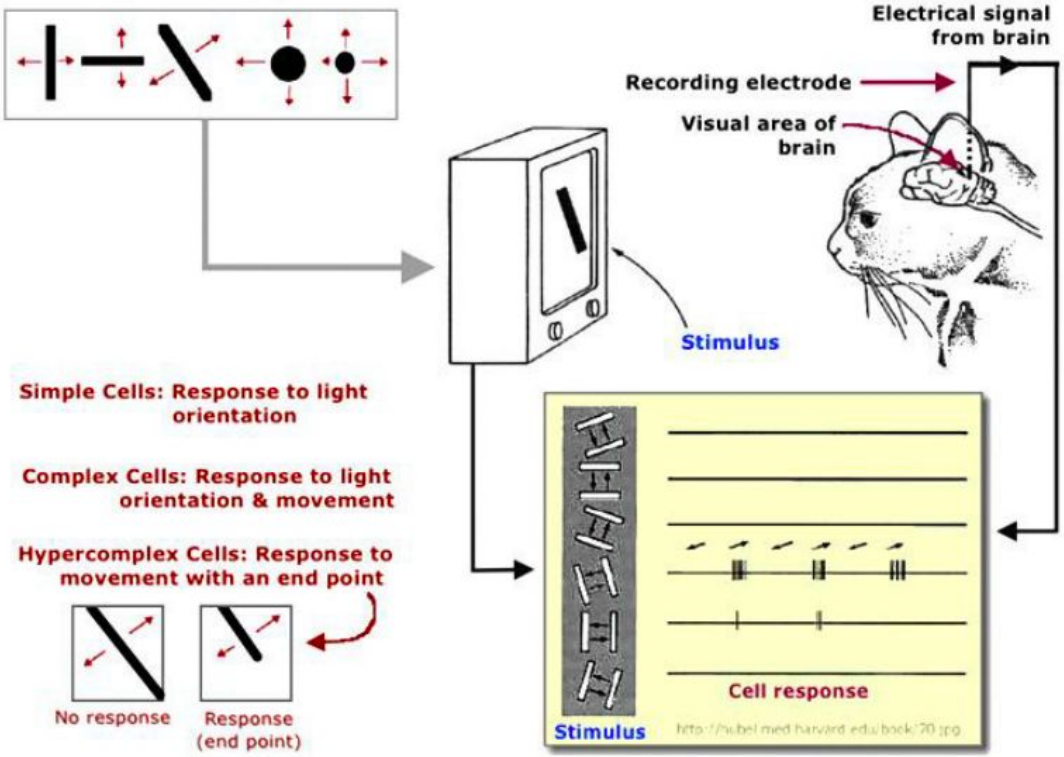
⇒ Brains extract features by neurons.

The work of David Hubel and Swede Torsten Wiesel gave clues to how a brain sees images from eyes.

Hubel and Wiesel Cat Experiment 1

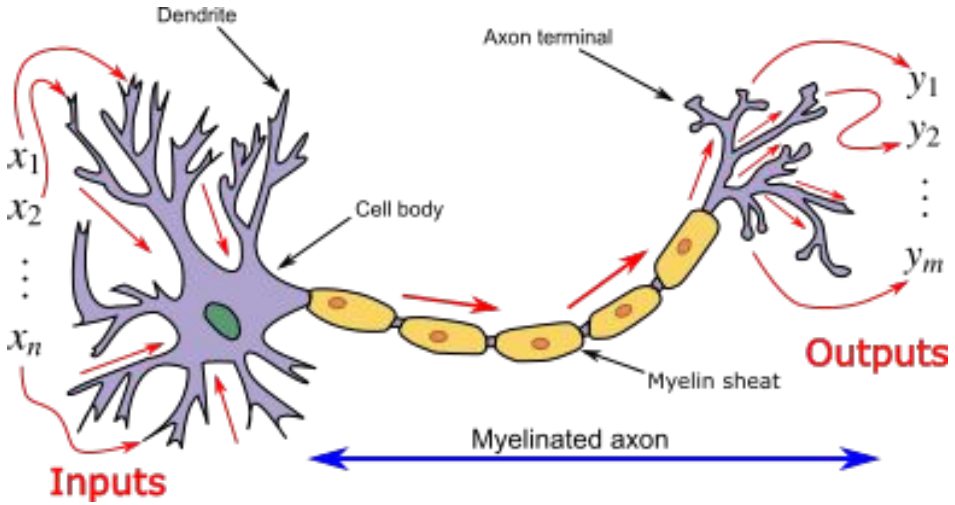
Hubel and Wiesel's experiment

Hubel and Wiesel Cat Experiment 2

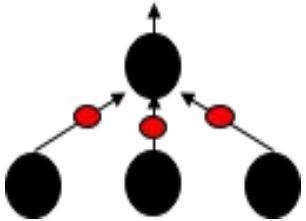


- a small microelectrode is jabbed into a brain cell in the visual cortex at the back of the brain of an anesthetized cat.
- Some neurons fire rapidly when presented with lines at one angle, while others respond to another angle.
- It shows that neurons generate different signals for different observations

Anatomy of neurons and artificial neurons



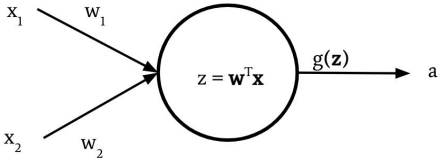
Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals



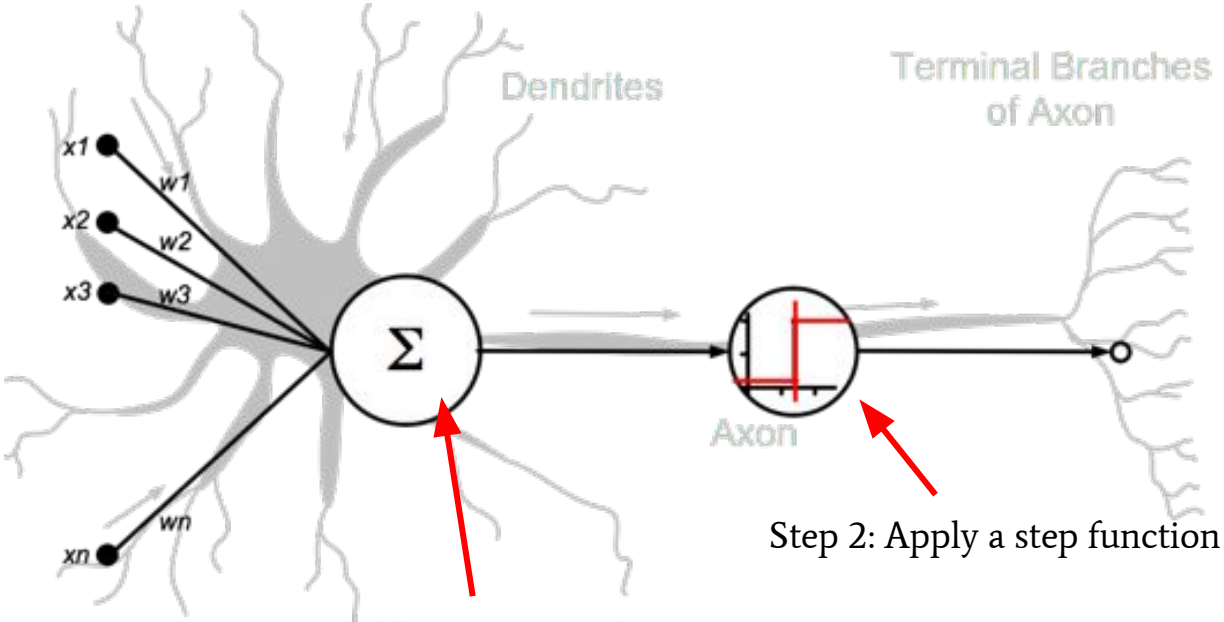
An artificial neuron:

- Receive input
- Change the internal state (activation) according to that input
- Produce output depending on the input and activation

Modelling the neuron



A perceptron



Step 1. Adds up weighted sum

Step 2: Apply a step function

Integrate-and-fire

Forming a network

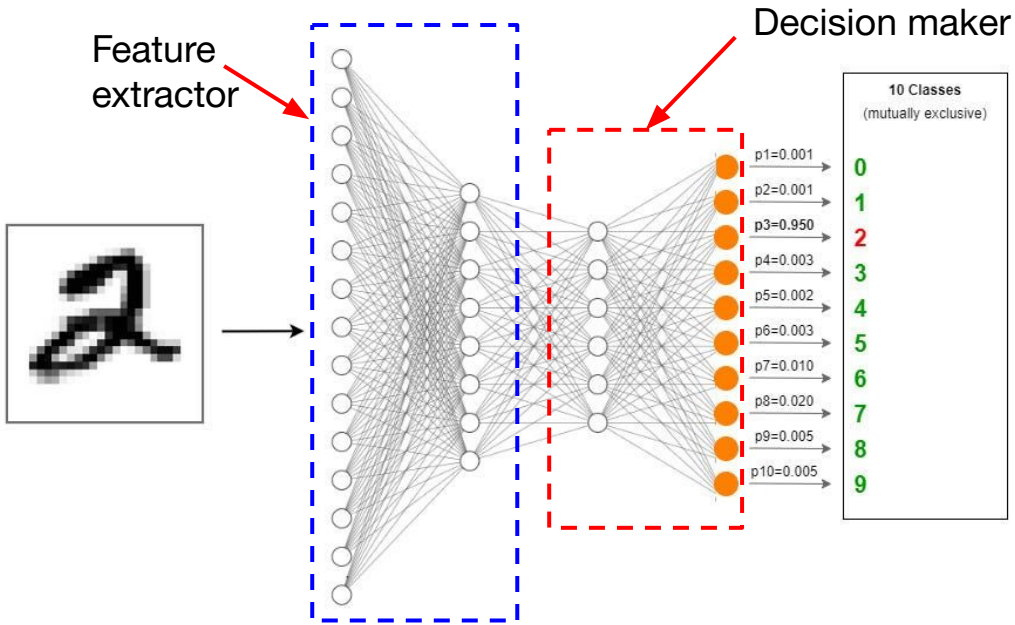
A number of connected neurons make up a network. (Connectionism)

Neurons (~10^{11}) → Circuity (~10^{4}) → Brain

(Brain costs only 25W; a huge energy-efficient device)

Feature extraction layers + decision making layer

- A neural network can be considered as a stack of neuron layers
- The first several layers play the role of a feature extractor; the last layers are the decision maker



Neural network from a mathematical perspective

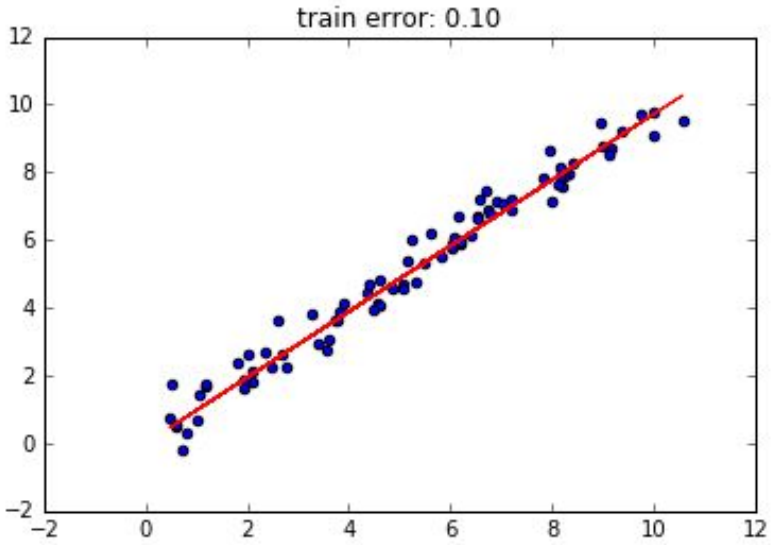
- It looks like a Matryoshka (Russian nesting doll)!

Linear Models for Regression

If the relationship between y and x is linear, we can fit the (x, y) by a line,

$$y = wx + w_0$$

We can learn w by gradient descent.



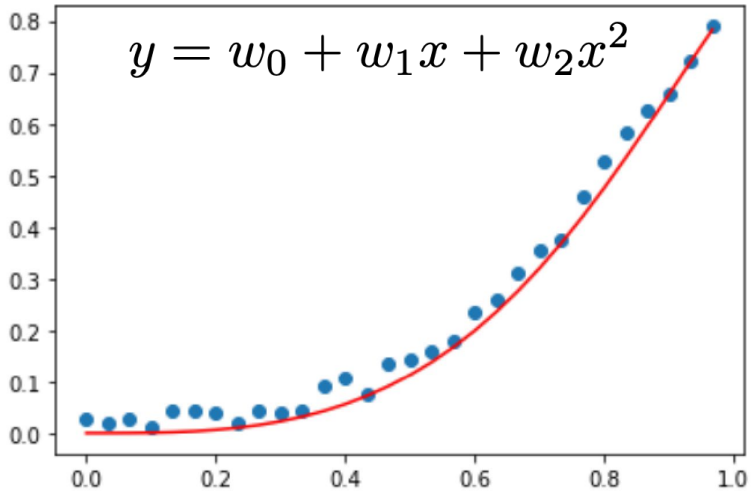
Polynomial Models for Regression

If the relationship is nonlinear, we can fit the (x, y) by curves, for example, a polynomial.

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + ... + ...w_n x^n$$

$$= w_0 + \sum_{i=1}^{n} w_i x^i$$

$$= \sum_{i=0}^{n} w_i x^i \text{ (here, we let } x^0 = 1)$$



- Actually, it is a generalized form of the linear model. If w_2 … w_n are all 0s, it reduces to a linear model.

Generalizing the model

Let's generalize the regression model so it can fit a complicated dataset.

- There are some changes we can do with the polynomial model,
  - Let x be a vector **X:** $x \rightarrow \mathbf{x} = (x_1, x_2, ..., x_D)$
  - Replacing polynomials by a set of base functions $\{\phi_i(\mathbf{x})\}$
  - Let y be a function of the sum of $\phi_i(\mathbf{x})$, so we have,

$$y = f(w_1\phi_1(\mathbf{x}) + ... + w_N\phi_N(\mathbf{x}) + w_0)$$

$$= f(\sum_{j=1}^{N} w_j\phi_j(\mathbf{x}) + w_0)$$

$$= f(\sum_{j=0}^{N} w_j\phi_j(\mathbf{x}))$$

This form is more adaptive than the polynomial.

- ϕ can be any functions
- f decides the output y
  - If f is piecewise function ⇒ y is discrete
  - If f is continuous function ⇒ y is continuous

Actually, the form depicts how people understand data.

From generalized regression model to neural network

We define $\phi_j(\mathbf{x}) = g_j(z_j)$, where

- $z_j = w_{j1}x_1 + w_{j2}x_2 + ... + w_{jD}x_D + w_{j0} = \sum_{i=1}^{D} w_{ji}x_i + w_{j0}$.
- g_j is a nonlinear function; we call it as activation function.

Let $a_j = g(z_j)$, we have

The same form

- $y = f(\sum_{j=1}^{N} w_j a_j + w_0) = f(\sum_{j=1}^{N} w_j g_j(\sum_{i=1}^{D} w_{ji}x_i + w_{j0}) + w_0)$
- if we replace the x_i with $\psi(x_i)$, a function like ϕ_j, the nest goes much deeper!

With the ϕ we defined, we have a model that is highly adaptive, but also it looks like a Matryoshka (Russian nesting doll)!

From generalized regression model to neural network

We have a generalized model $y = f(\sum_{j=0}^{N} w_j \phi_j(\mathbf{x}))$. Let $\phi_j(\mathbf{x}) = g_j(z_j)$, it becomes the math form of a neural network.

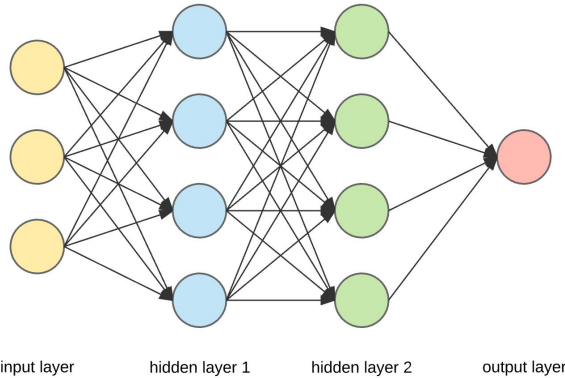
Their connections are in the

- accumulation z_j
- activation function g_i

Neurons behave like this.

- the nesting structure of the form

why ?



Mathematically,

- z_j is a linear function that sums up elements in **x**.
- g_j is a nonlinear function that changes the correlation of the variables

Intelligence can be a two-step process

- Step 1: feature extraction, e.g., seeing, hearing, etc.
- Step 2: cognition, e.g., recognition, decision making, etc.

Recall the model, $$y = f(\sum_{j=0}^{N} w_j \phi_j(\mathbf{x}))$$

- ϕ is like the feature extraction,
- f is like the cognition

This math form describes how “intelligence” works.

Neural Network Summary

A neural network is a particular case of the generalized regression model $y = f(\sum_{j=0}^{N} w_j \phi_j(\mathbf{x}))$.

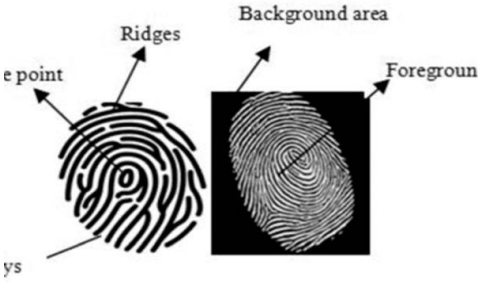
- We define $\phi_j(\mathbf{x}) = g_j(z_j)$, where z_j is a linear function (accumulation) and g_j is a nonlinear function (activation function).
- N is the number of neurons
- Adding layers is to replacing $\mathbf{x}$ with a function (i.e., increasing the number of nested functions)
- Weights (i.e., $\{w_j\}$) are learned by gradient descent with backpropagation.

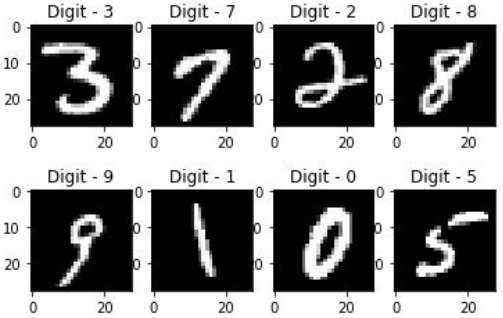
Agenda

- Artificial neural network
  - Perceptron and multilayer networks
- Understanding the neural network
  - from a biological perspective
  - from a mathematical perspective
- **Deep learning**
  - Convolutional neural network
  - Applications: style transfer (in appendix)

Applications of MLP

In 1990s, neural networks were highly developed. They were widely used in banks for fingerprint and handwriting recognition.





However, the applications were limited in these simple tasks,

- Complex tasks need more neurons and layers ⇒ the MLP structure results in an explosion of weights
- hardware limitations ⇒ we didn't have CPUs that could update the weight efficiently

Deep learning

- More neurons and layers (going deeper) ⇒ better features
- Going deeper ⇒ (exponentially) increasing number of weights ⇒ deep MLPs were inapplicable (due to the limits of memory size and computing power)

The breakthrough:

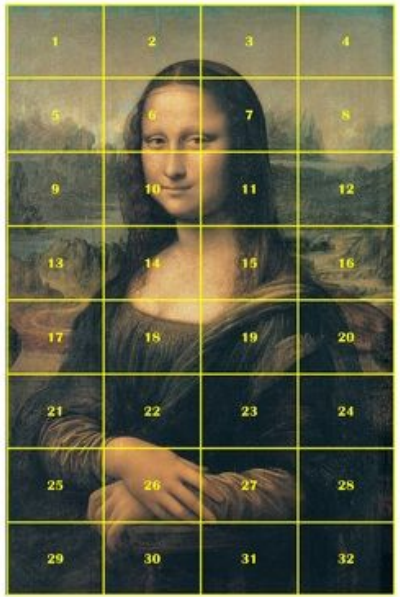
- The success of GPU computing ⇒ calculating thousands of weights simultaneously
- The development of convolutional neural networks (CNN) ⇒ building deep networks with (relatively) less weights

Convolutional neural network (CNN)

CNNs are a special type of neural network for processing data that has a known grid-like topology.

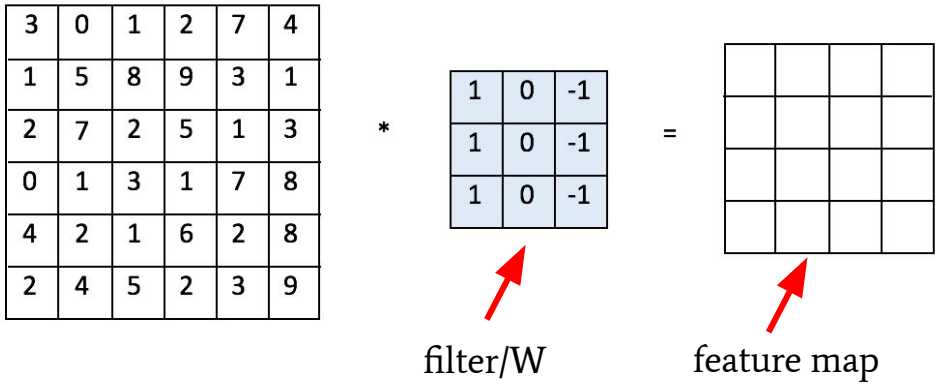
- Grid-like topology data: data that measured at regular time intervals, e.g., time series, and images (i.e., pixels of a 2-D grid)

Convolution: a mathematical operation defined on grid-like data;(also, it can be considered as a pattern for connecting neurons.)



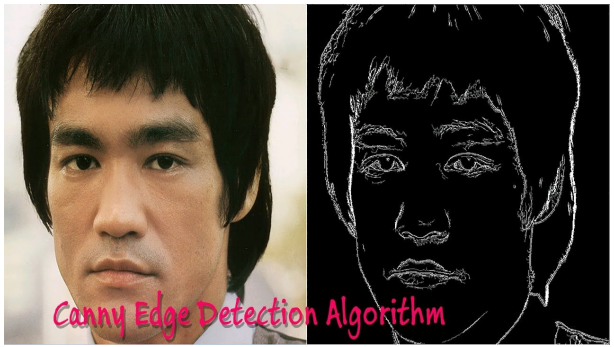
The convolution

Convolution is a kind of “accumulation”, which sums up the values in a particular region with a set of weights $\{w_i\}$ (called filter or kernel).

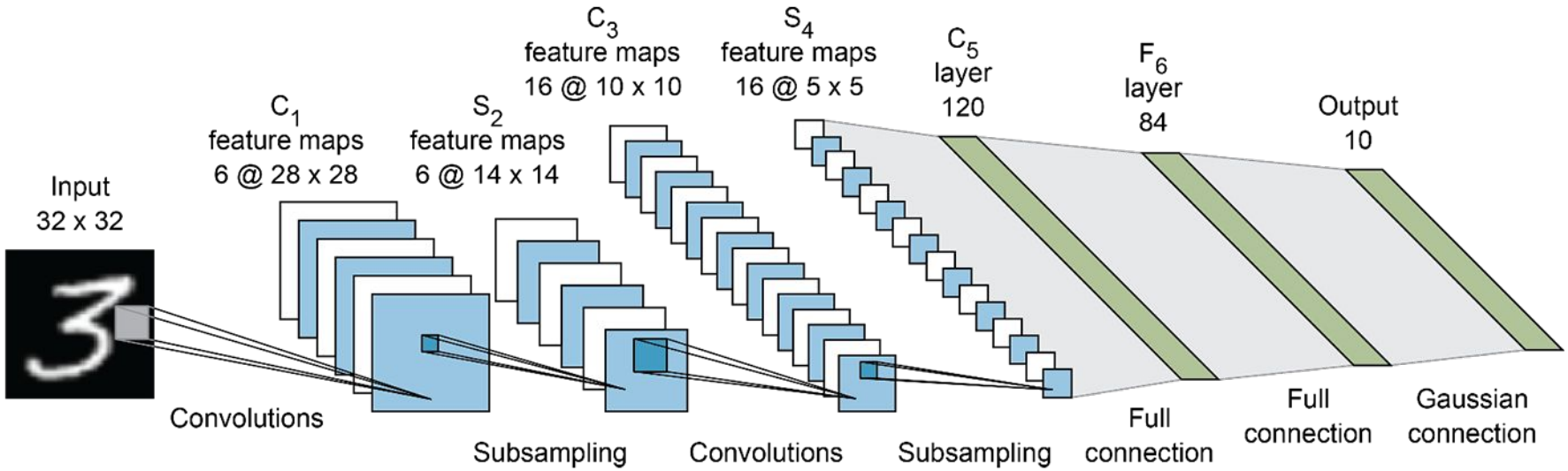


The filters are feature detectors.

- They can detect edges as the pixel values of two regions often varies dramatically.



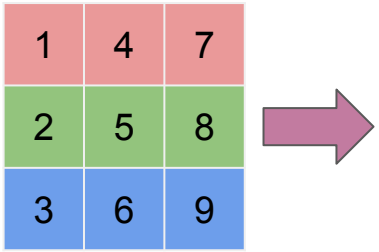
LeNet-5 architecture



- The input is an m x n array (e.g., a digital image)
- Layers (i.e., feature maps) are connected by convolution operators; feature maps is like “nodes” of the layer;
- C represents the convolutional layer; S represents subsampling (Pooling); F is the fully connected layer;
- The subsampling is like the activation function.

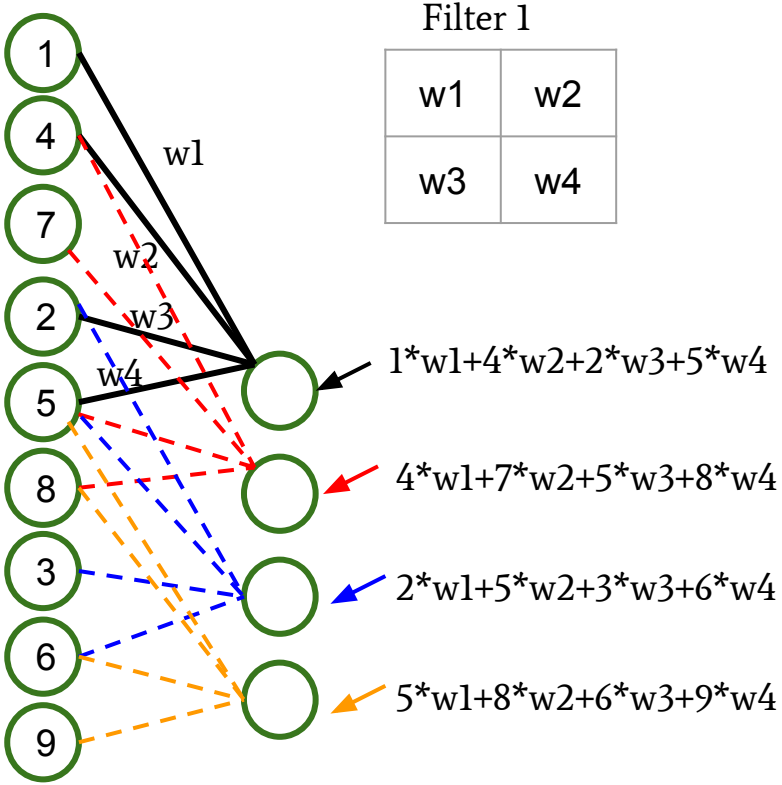
CNN is a sparsely connected MLP

A 2-D array (feature map) can be reshaped into an 1-D vector.



So, the convolution is equal to a sparsely connected network

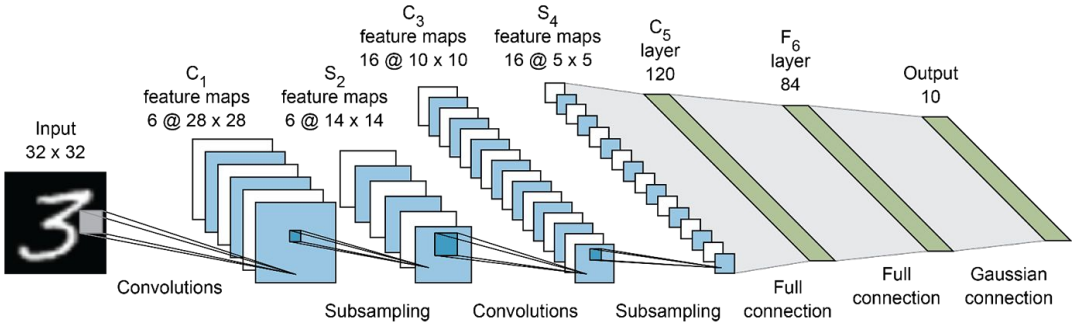
If we convolve the array with filter 1, the corresponding network is shown in the right.

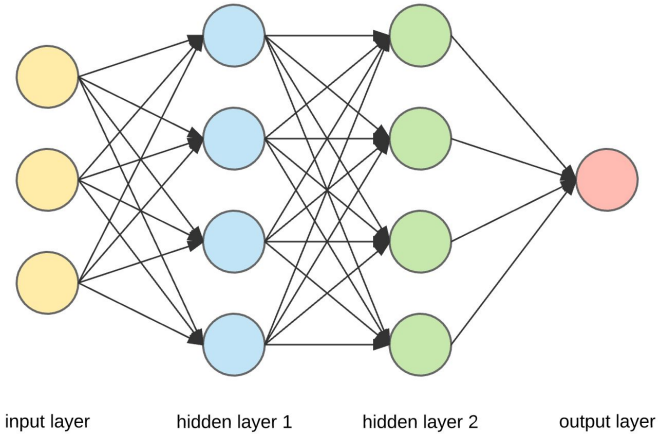


Filter 1

w1	w2
w3	w4

CNN v.s. MLP

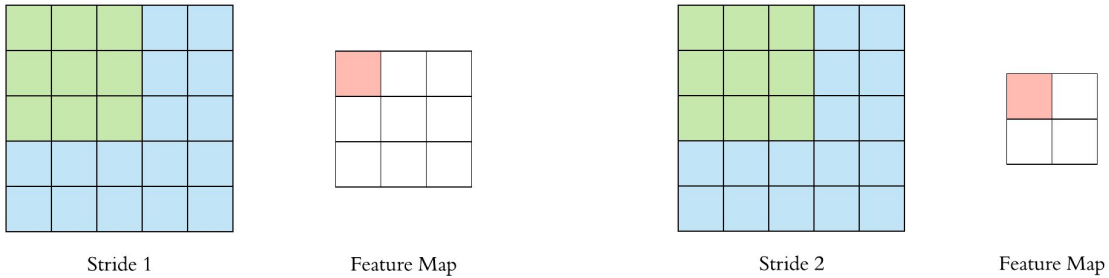




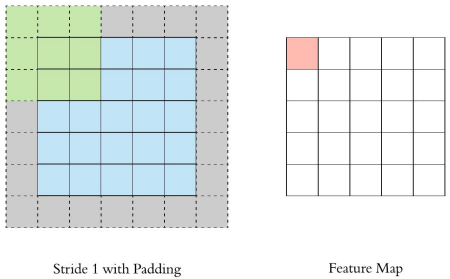
- Both contain multiple layers which allow each layer to find successively complex features.
- Connections in CNN is much sparser than MLP ⇒ CNN has less edge weights

CNN terms: stride and padding

Stride tells how much we move the filter at each step.



Padding, i.e., extending the inputs, allows a feature map has the same size as the inputs.



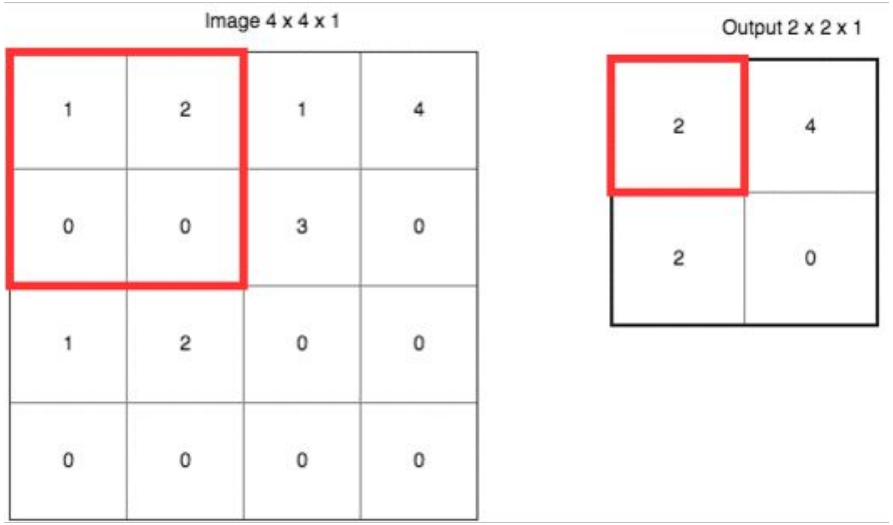
Zero-padding is often used.

- They are used to control the size of the output, which is important in practice.

The pooling (subsampling)

To remove the redundancy and reduce the size of the layer (and so, the number of parameters)

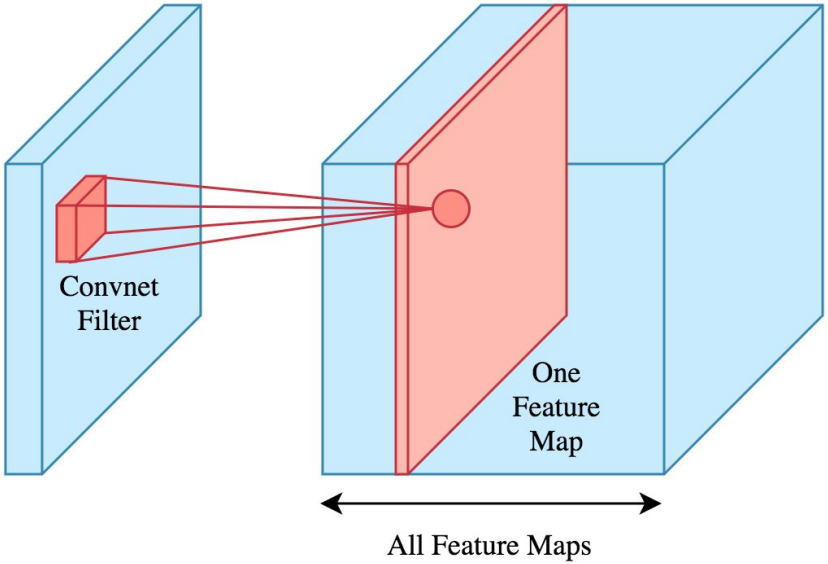
- Max pooling (good for image related tasks)
- Average pooling



Max Pooling

Feature maps

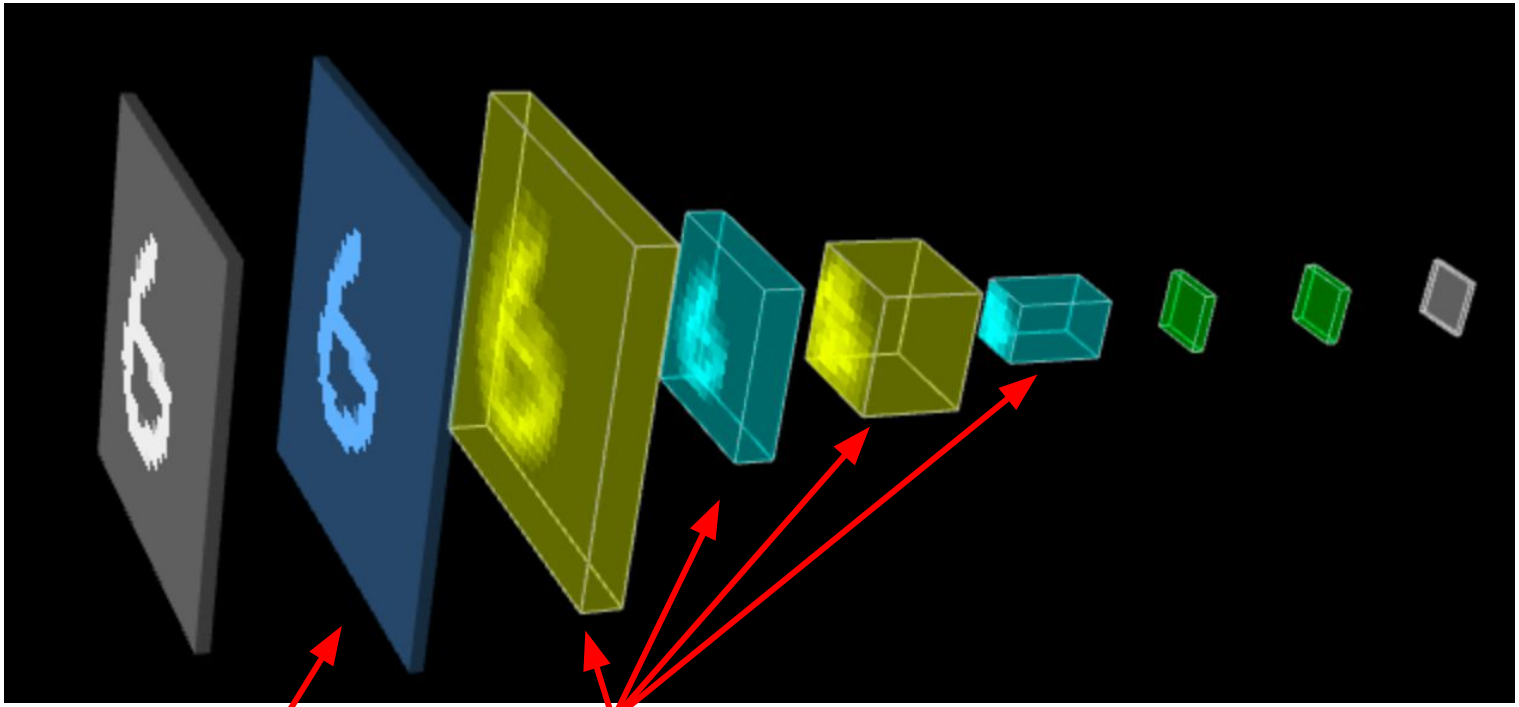
- They are the outputs of a kernel.



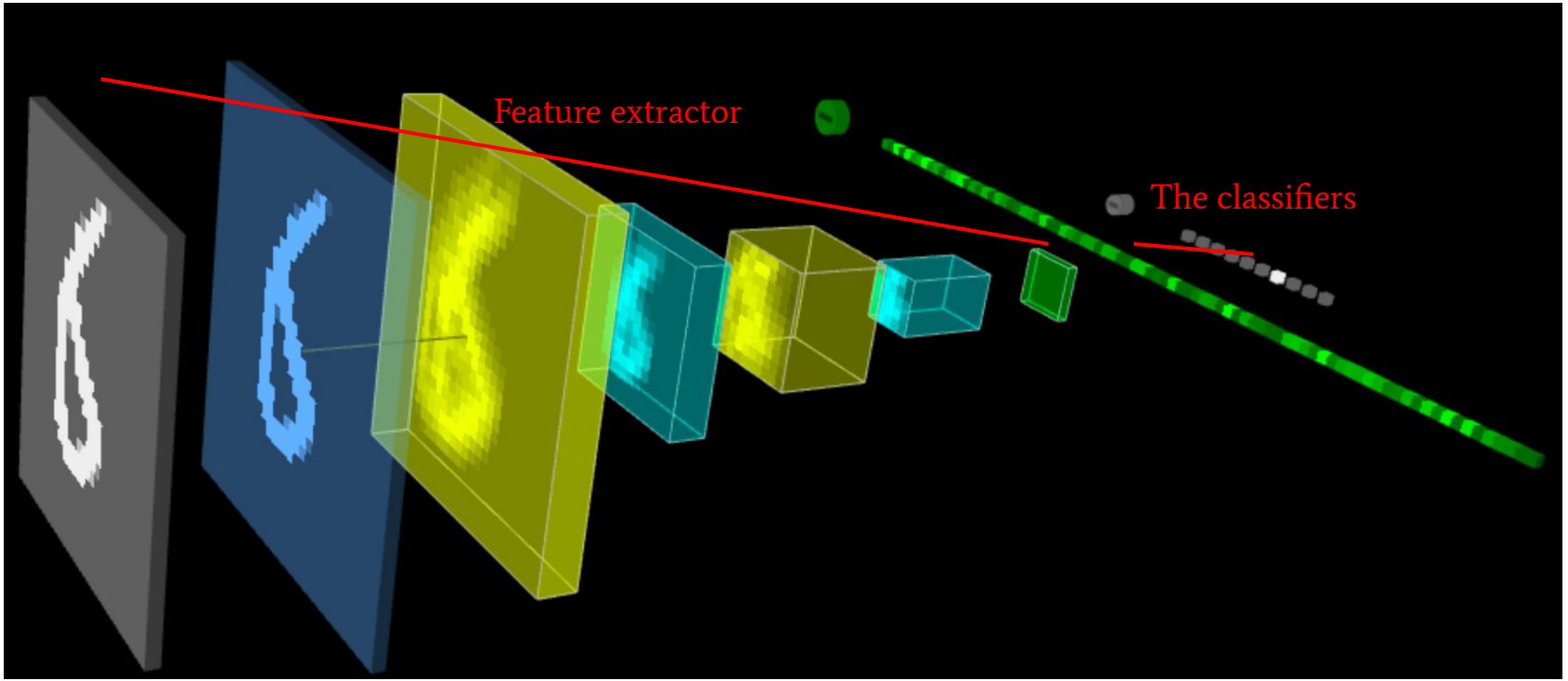
- It is always fun to visualize them.

Remark: Actually, there are different saying about what feature map is. Some people call matrices of pooling layers as feature map also.

An Online LeNet



An overview



Exercise: Build a CNN for the MNIST dataset

```
import torch.nn as nn
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(32 * 7 * 7, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x    # return x for visualization
```

PyTorch Convolutional Neural Network With MNIST Dataset | by Nutan | Medium

Google colab: https://colab.research.google.com/drive/12GQzpjzWXWDWqxCWsQLT9AtMhV0qEmnU?usp=sharing (please visit it with your nyu account)

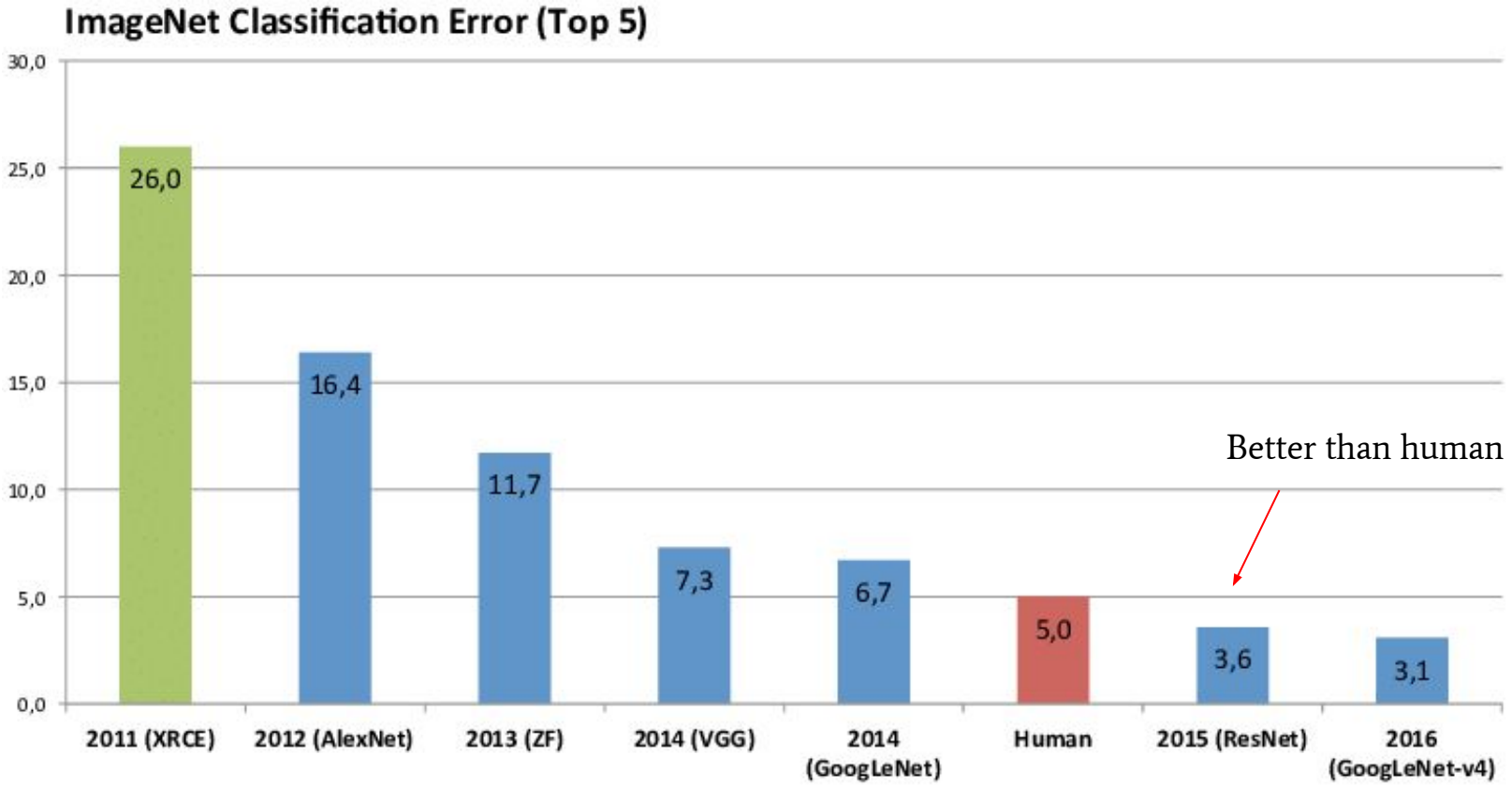
- Free GPU access
- Better GPU for subscribed users

Homework:

- Run through the notebooks
- Which model will have better accuracy? MLP or CNN

Image recognition is an important task

Deep neural network is a milestone of image recognition



Appendix: Image Style Transfer

Your Turn

How did this happen!?

⇒ Hint: human eyes perceive both low- & high-level features

⇒ What does low- & high-level features mean to computer vision

⇒ What are the low- & high-level features in CNN?

Discuss your thoughts with your class neighbors.

Scan to answer!

It even works for the video.

That paper aroused the research in AI & Arts.

Neural style transfer

Leon A. Gatys et al., Image style transfer using convolutional neural networks, CVPR 2016

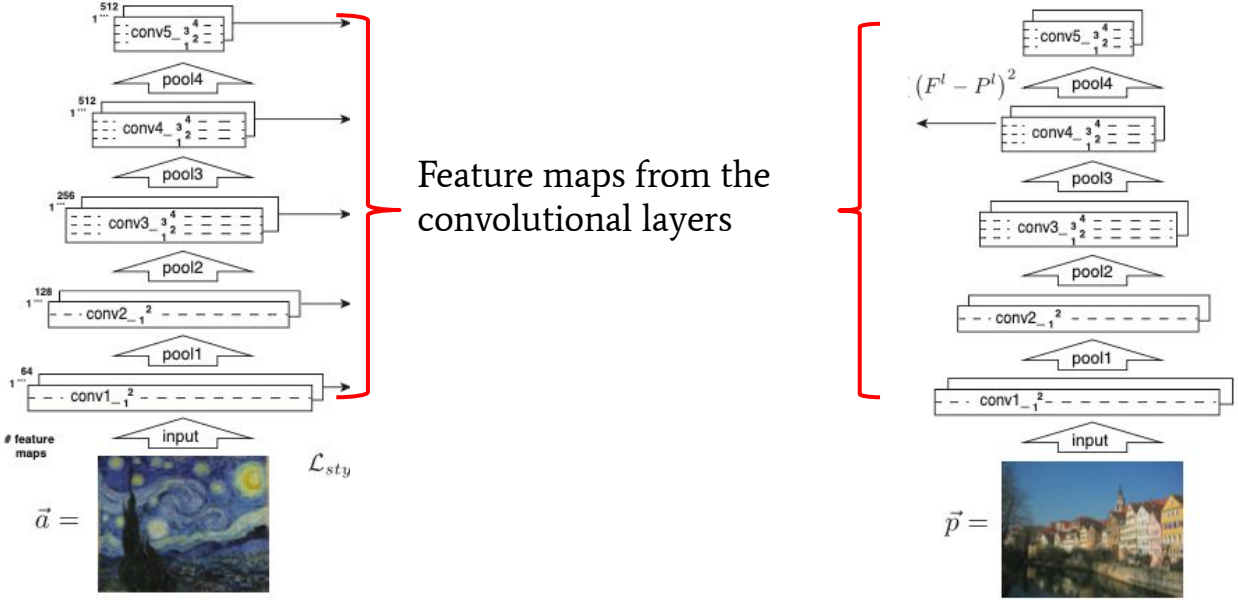
Digital images are just groups of pixel values. How can we copy the "style"?

- Or, we should ask first: What feature can be used to represent "style"?

  They use feature maps.

Neural style transfer

Input an image into an pre-trained ConvNet, we get the feature maps of the image.



The Features

Define content feature and style feature based on the feature maps.

- The content:

  For a given input image $\mathbf{p}$, we use a matrix $P_{i,j}^{l} \in R^{N_l \times M_l}$ to represent the feature map of the i^{th} filter at position j in layer l. So, the content of $\mathbf{x}$ is represented by a set of P.

- The style:

  For a given input image $\mathbf{a}$, we use a Gram matrix $G^{l} \in R^{N_l \times N_l}$ to represent its style, where G_{ij}^{l} is the inner product of the feature map of the layer l.

Loss function

Define the loss function. Let p, a and x represent the photo, artistic painting and the generated image respectively.

- The content loss at layer l:

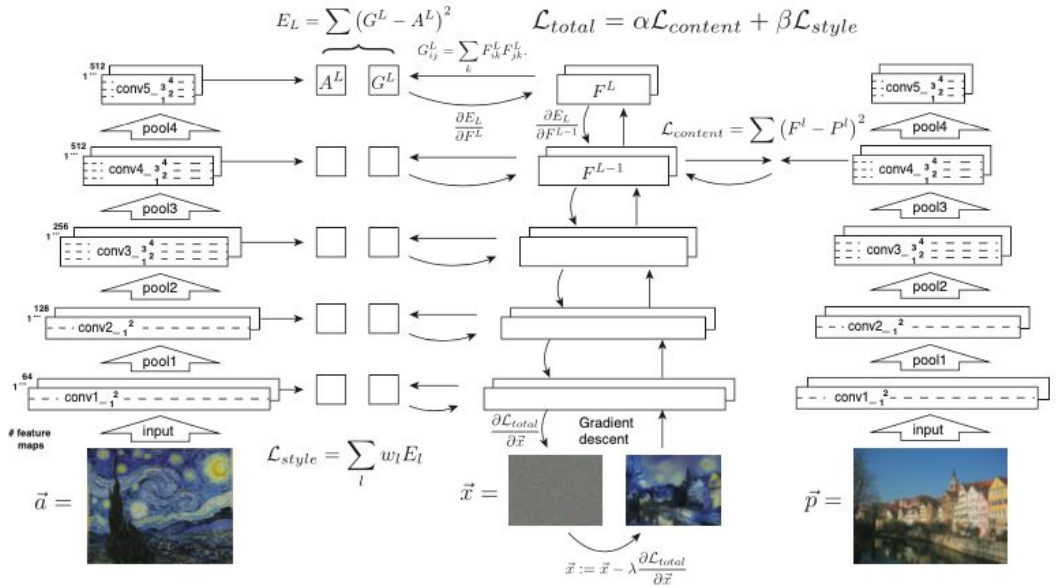
$$L_{content}(\mathbf{p}, \mathbf{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

- The style loss of layer l:

$$E_L = \frac{1}{4N_L^2 M_l^2} \sum_{i,j} (G_{i,j}^l - A_{i,j}^l)^2$$

Minimizing the loss

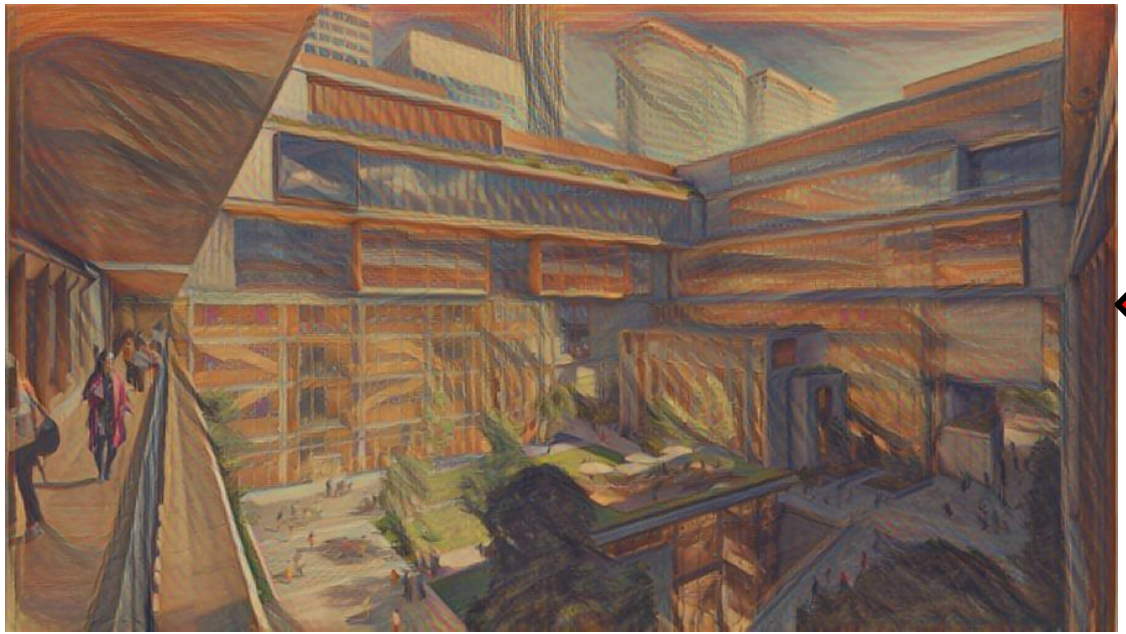
We then have the total loss (i.e., $L_{content} + L_{style}$). Minimizing the total loss, we get a “fused” image x.

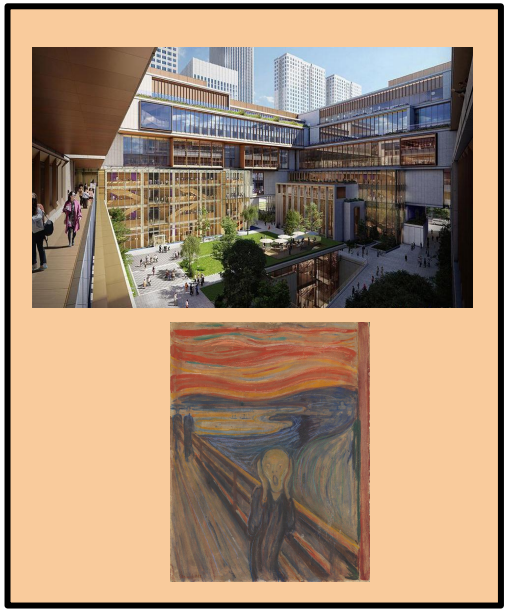


- Input the oil painting⇒take the feature maps ⇒compute the style feature
- Input the photo⇒take the feature maps⇒compute the content feature
- Input an image x of white noise⇒take the feature maps⇒compute the style feature and content feature
- Compute the total loss⇒minimize the total loss ⇒ but, instead of changing w, we change the pixel values in x !

Now try some photos at:

https://style-transfer.makeoptim.com/





Stable diffusion algorithm

You may try this link : https://gencraft.com/generate

This algorithm allows you to input text prompts, and it will generate images accordingly.