CS5242 : Neural Networks and Deep Learning

Lecture 6: Multi-Layer Perceptron
Depth

Semester 1 2021/22

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

Outline

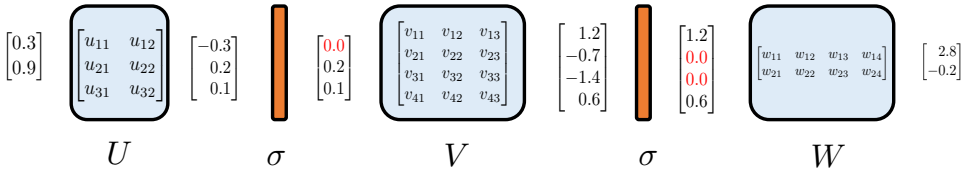
- The three representations of neural networks
- The importance of non-linearities
- The importance of multiple layers
- Labs with MNIST and CIFAR

Outline

- The three representations of neural networks
- The importance of non-linearities
- The importance of multiple layers
- Labs with MNIST and CIFAR

The three representations

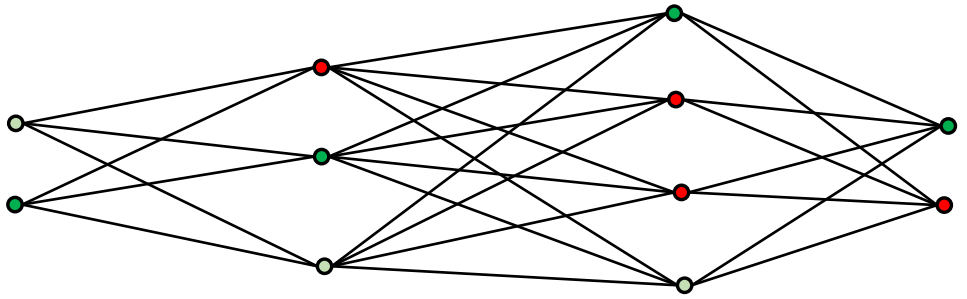
Computer Science



Mathematics

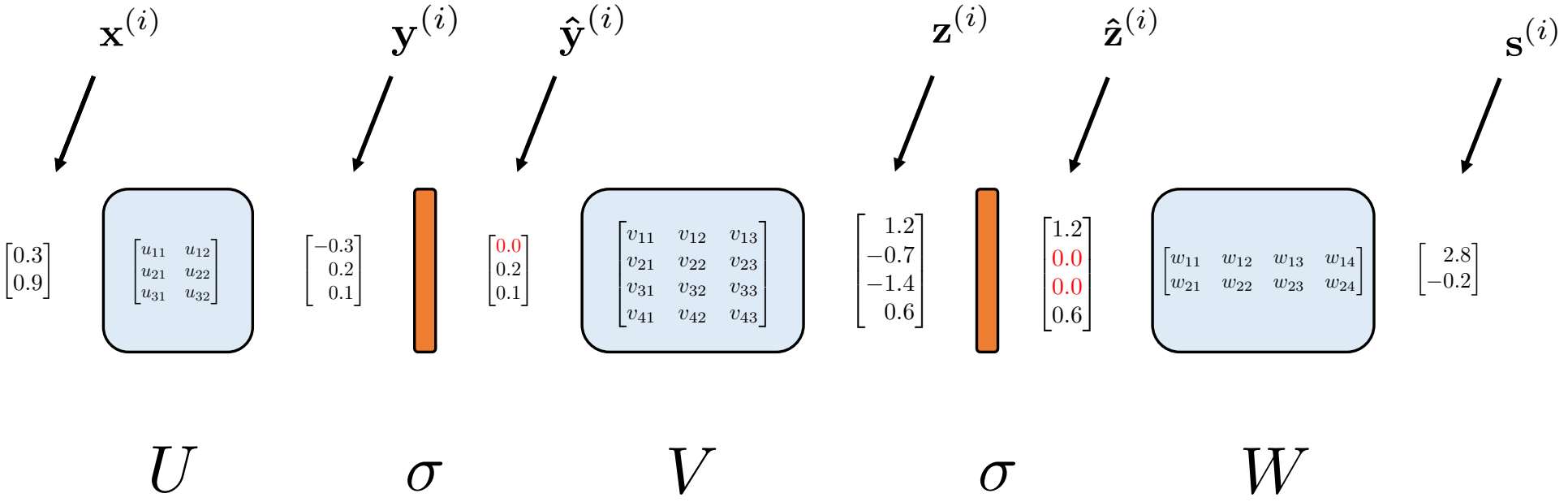
$$\mathbf{s^{(i)}} = W \ \ \sigma(V \ \ \sigma(U\mathbf{x^{(i)}}) \ \ )$$

Biology



Computer science representation

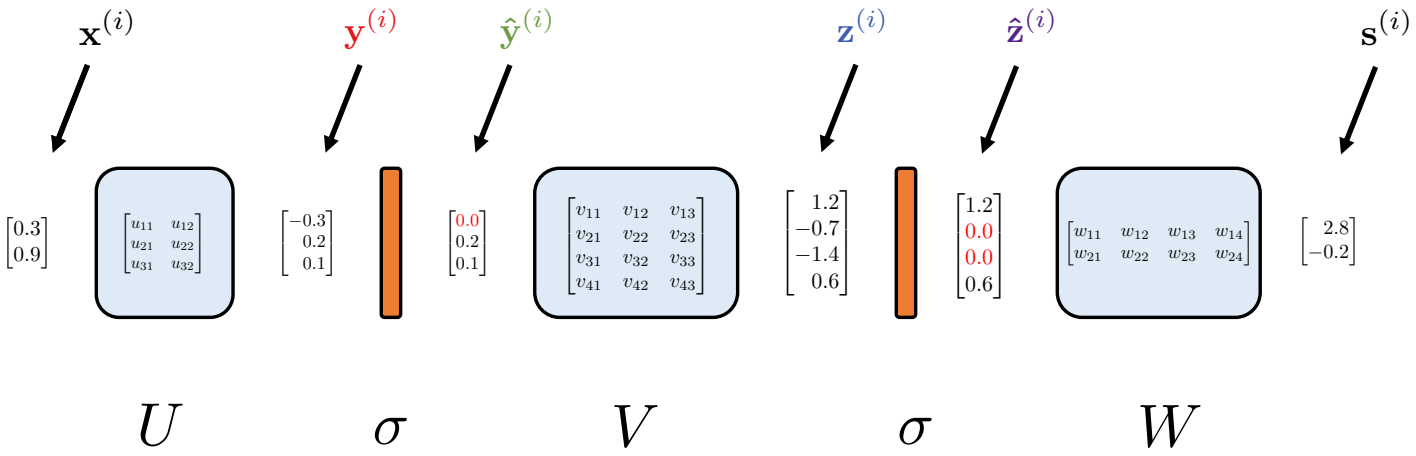
- Computer science visualization of a neural network:



Mathematical representation

- Mathematical formulation of a neural network :

$$\mathbf{s}^{(i)} = W\ \sigma\ (V\ \sigma\ (U\mathbf{x}^{(i)}\ )\ )$$

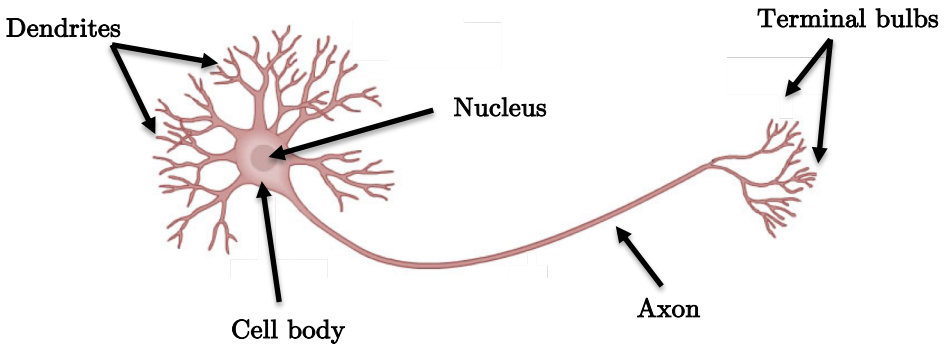


$$\mathbf{y}^{(i)}$$
$$\hat{\mathbf{y}}^{(i)}$$
$$\mathbf{z}^{(i)}$$
$$\hat{\mathbf{z}}^{(i)}$$
$$\mathbf{s}^{(i)}$$

A neural net is simply a composition of linear and nonlinear functions !

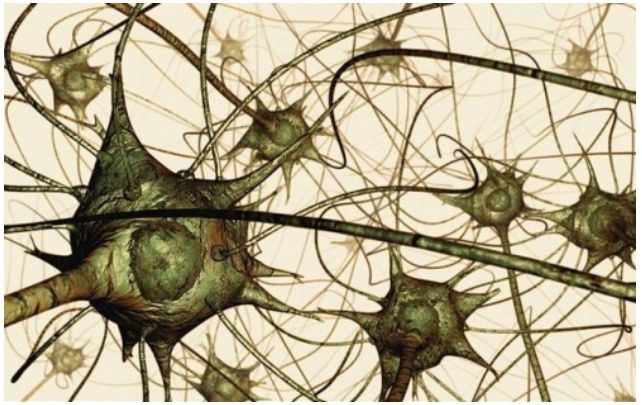
Biological representation

- A basic neuron:



- Brain = neural networks

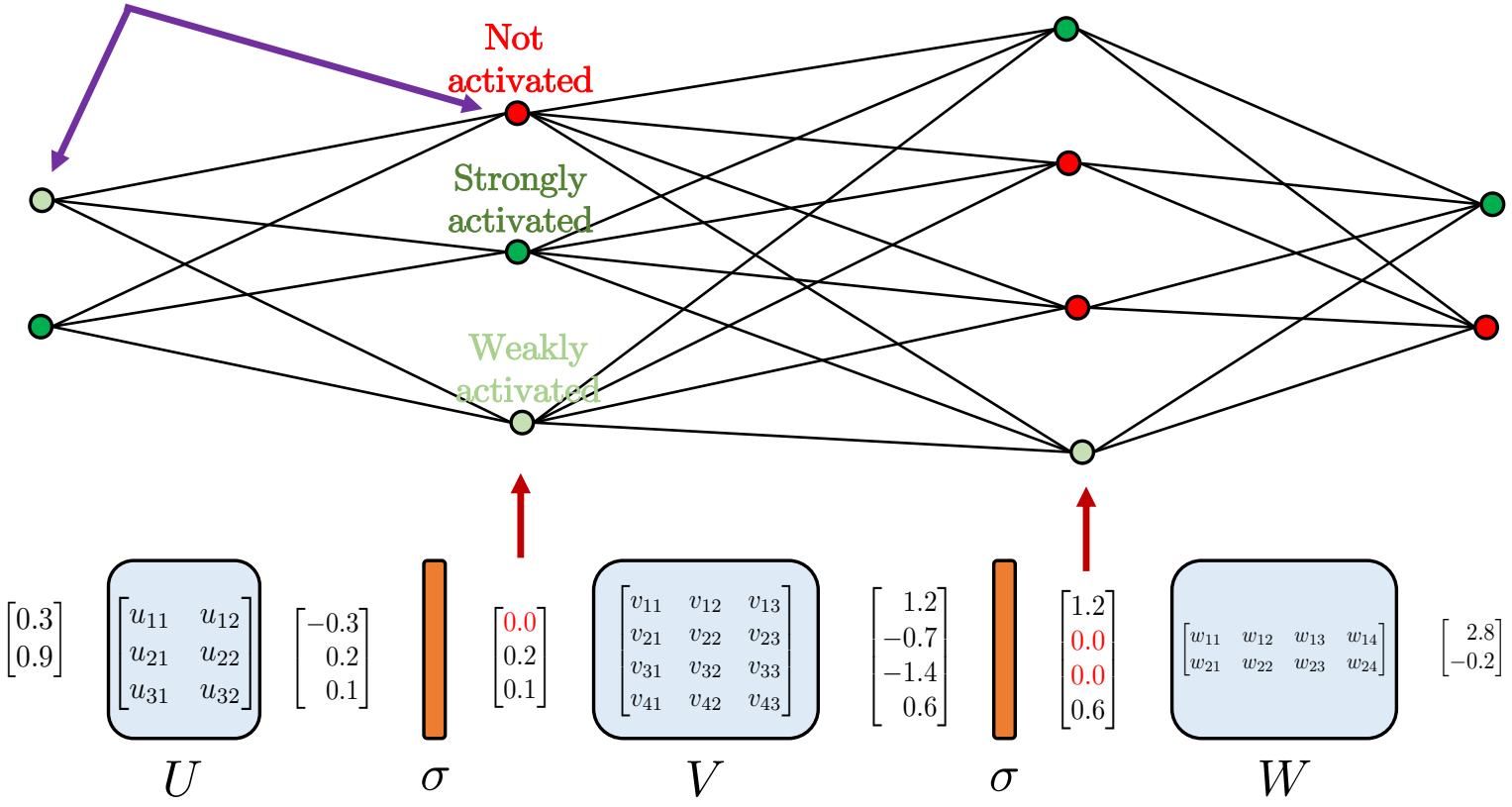
(group of connected neurons)



- A human brain has 10^{11} neurons and 10^{14} axons.

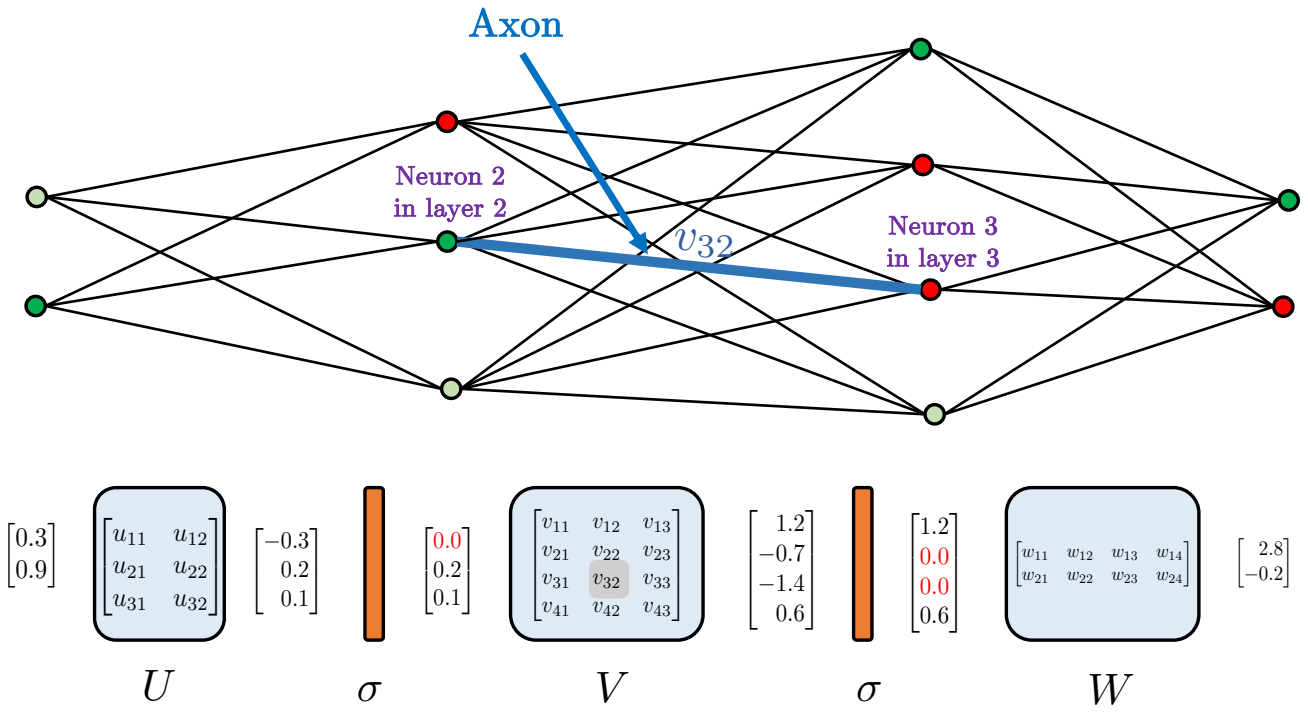
Biological representation

- Biological formulation of a neural network :
  - Neurons



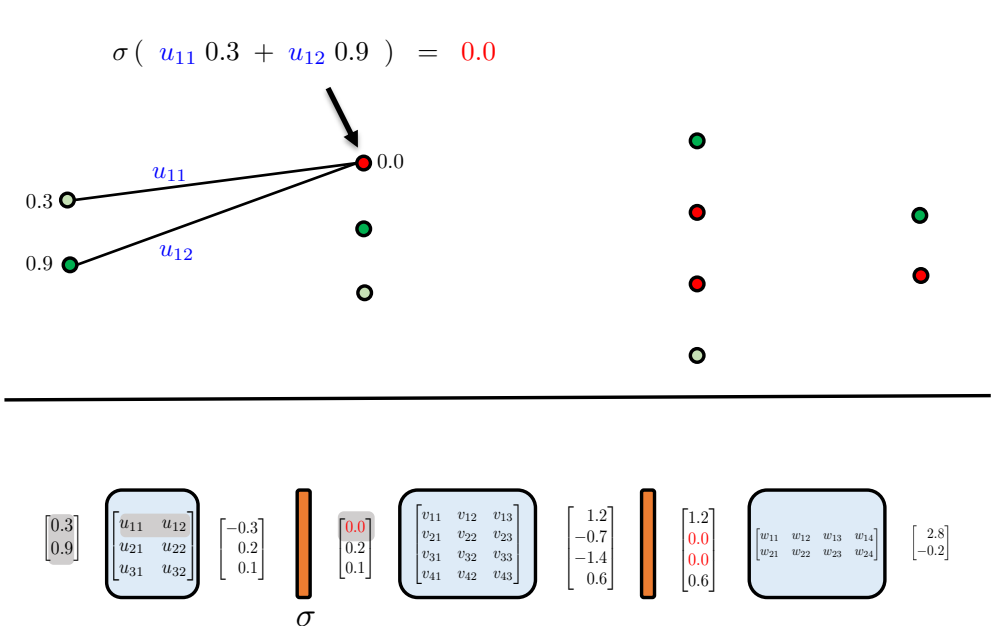
Biological representation

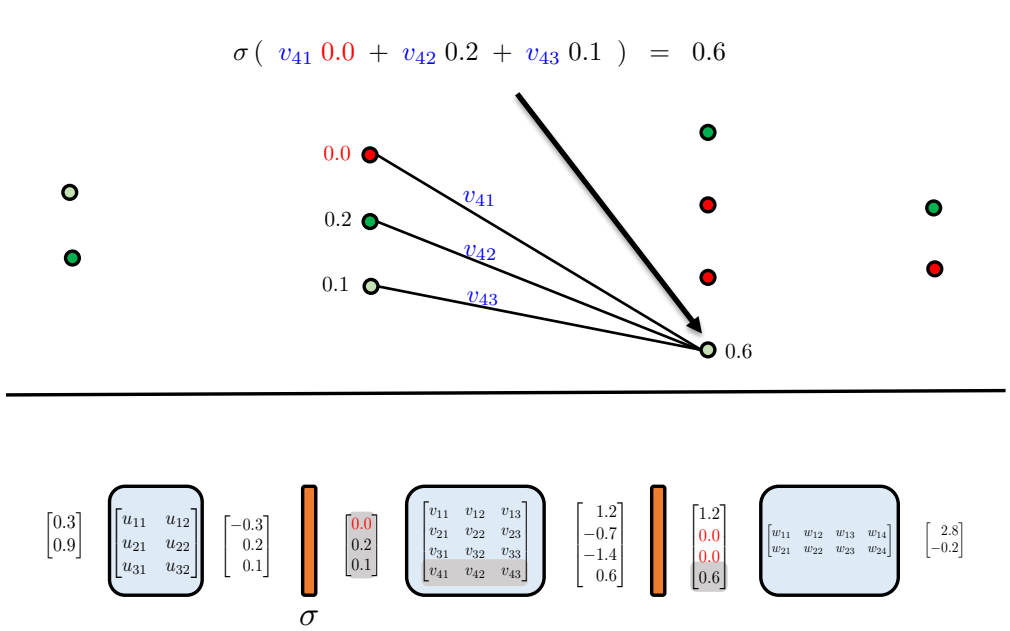
- Axons :
  - Each weight in the network can be thought as the strength of a neuronal connection.
  - Weight v_{32} = strength of the connection between neuron 2 in layer 2 and neuron 3 in layer 3.



Biological representation

- Biological representation of an activation :

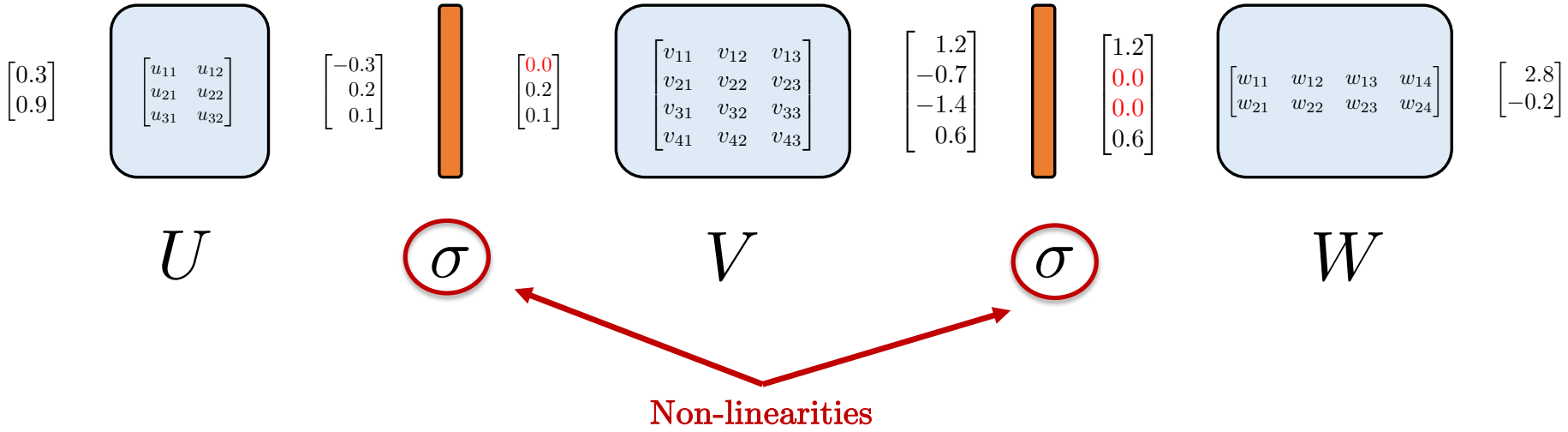




Outline

- The three representations of neural networks
- **The importance of non-linearities**
- The importance of multiple layers
- Labs with MNIST and CIFAR

The need of non-linearities



- Why non-linearities are introduced between layers?
  - Biological motivation
  - Mathematical motivation

Mathematical motivation

- Without non-linearities, the network degenerates as a linear model.

With nonlinearities:

$$\mathbf{s}^{(\mathbf{i})} = W\sigma(\ V\sigma(\ U\mathbf{x}^{(\mathbf{i})}\ )\ )$$

Without nonlinearities:

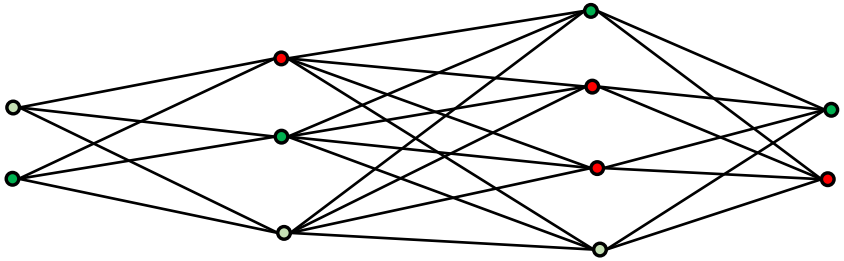
$$\mathbf{s}^{(\mathbf{i})} = \underbrace{W \quad V \quad U}_{A}\mathbf{x}^{(\mathbf{i})}$$

A product of matrices is just another matrix :

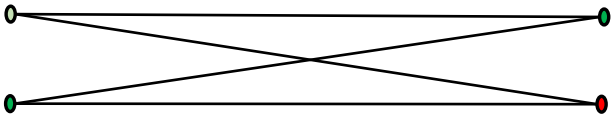
So the network collapses to a one-layer network !

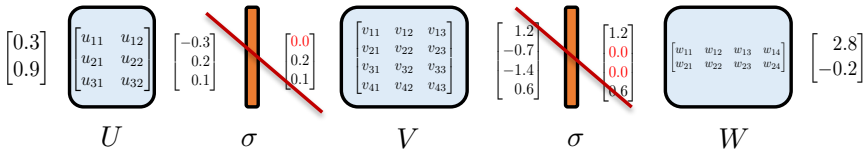
Mathematical motivation

- Without non-linearities:

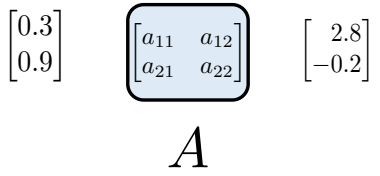


$\Rightarrow$





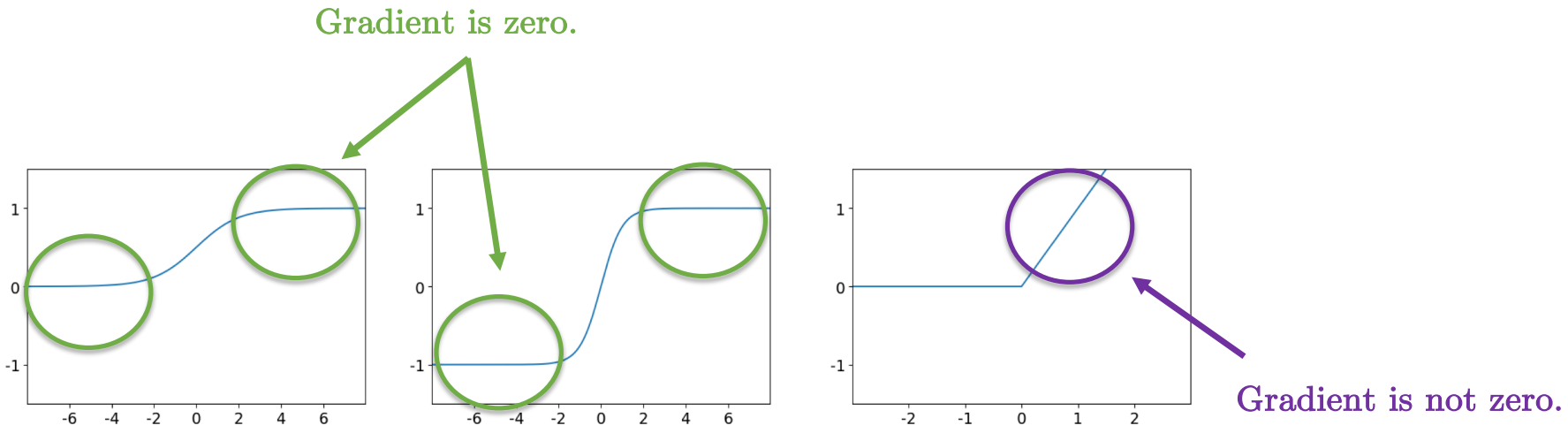
$\Rightarrow$



$$\mathbf{s^{(i)}} = W \cancel{\sigma(V \sigma(U\mathbf{x^{(i)}}))} \Rightarrow \mathbf{s^{(i)}} = A\mathbf{x^{(i)}}$$

Limitations of sigmoid and tanh

- Issues with Sigmoid and Tanh:
  - They are responsible for the vanishing gradient problem ⇒ Weights will not be updated.



Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Tanh

$$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

ReLU

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Outline

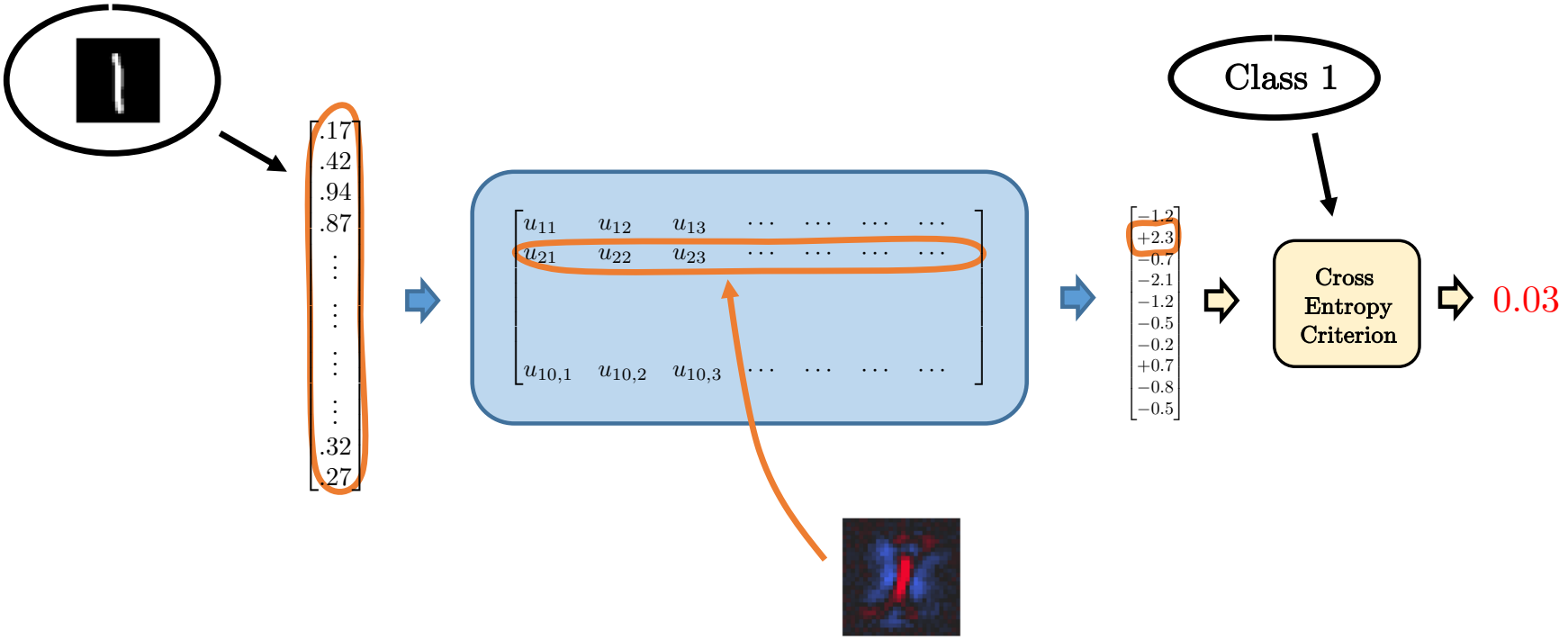
- The three representations of neural networks
- The importance of non-linearities
- **The importance of multiple layers**
- Labs with MNIST and CIFAR

An example

- Let us consider two classification problems to demonstrate the need of multiple layers :
  - An easy classification problem
  - A "hard" classification problem

An easy classification problem

- Classify the digit numbers 0,1,2...9 :
  - One-layer net with 10 classes is enough to solve this classification problem.

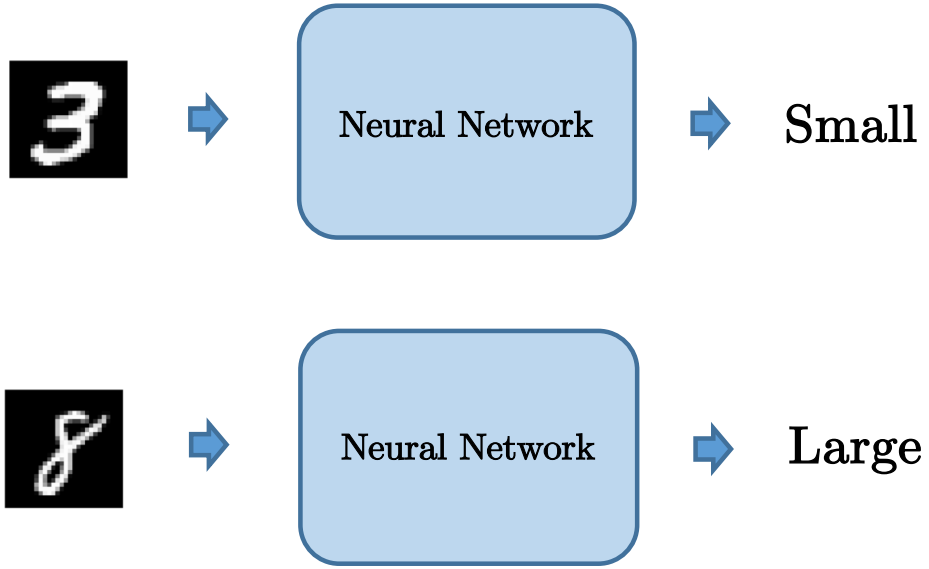


A hard classification problem

- A harder classification problem:
  - Classify small vs large digits into 2 categories

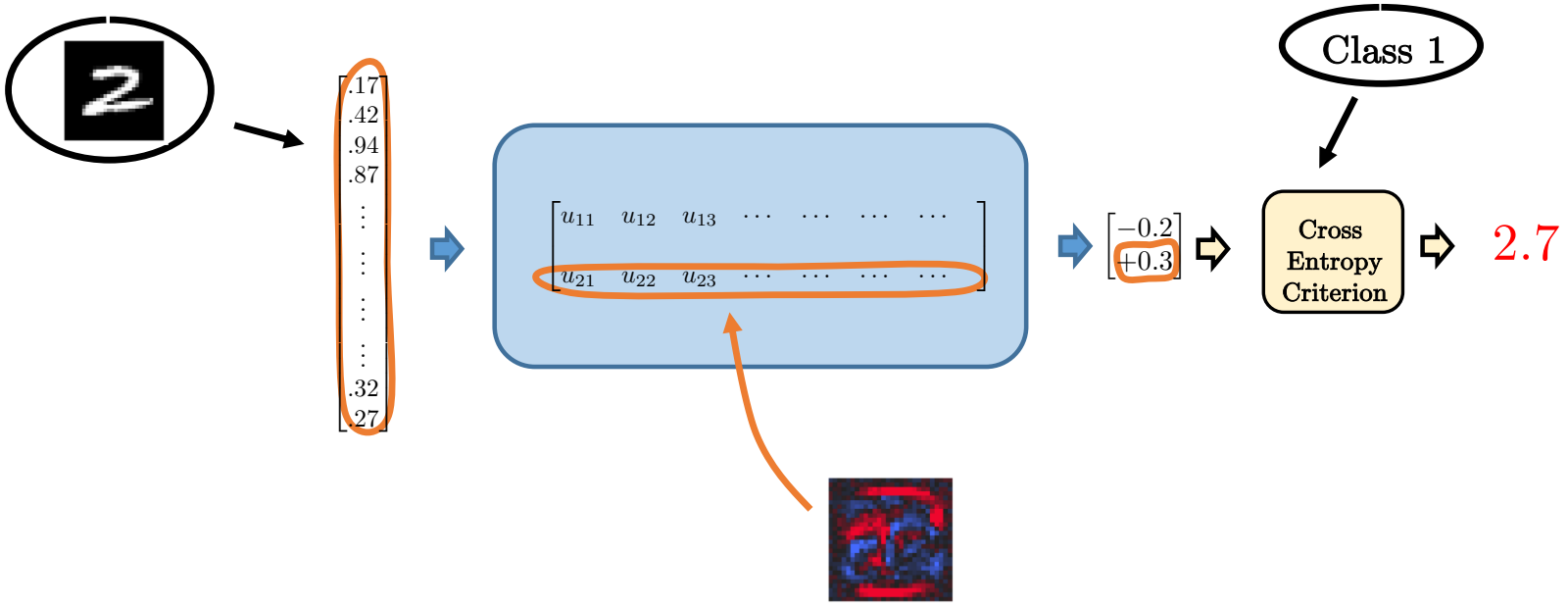
Class 1: Small (0,1,2,3,4)

Class 2: Large (5,6,7,8,9)



A hard classification problem

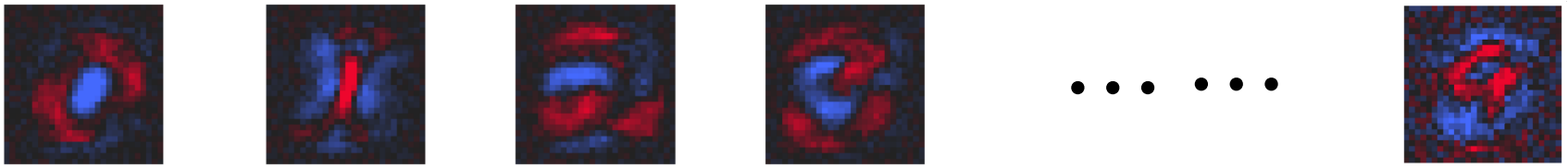
- Classify small vs large digits into 2 categories :
  - One-layer net with 2 classes is not enough to solve this classification problem.



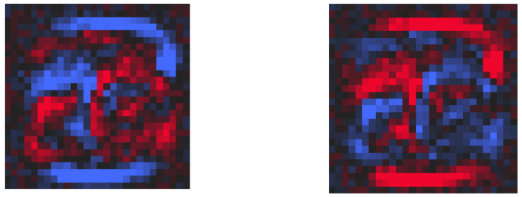
This pattern tries to represent digits 5-9 simultaneously with no success.

Template matching

- It is easy to find good templates for zero, one, two, three, ..., nine :

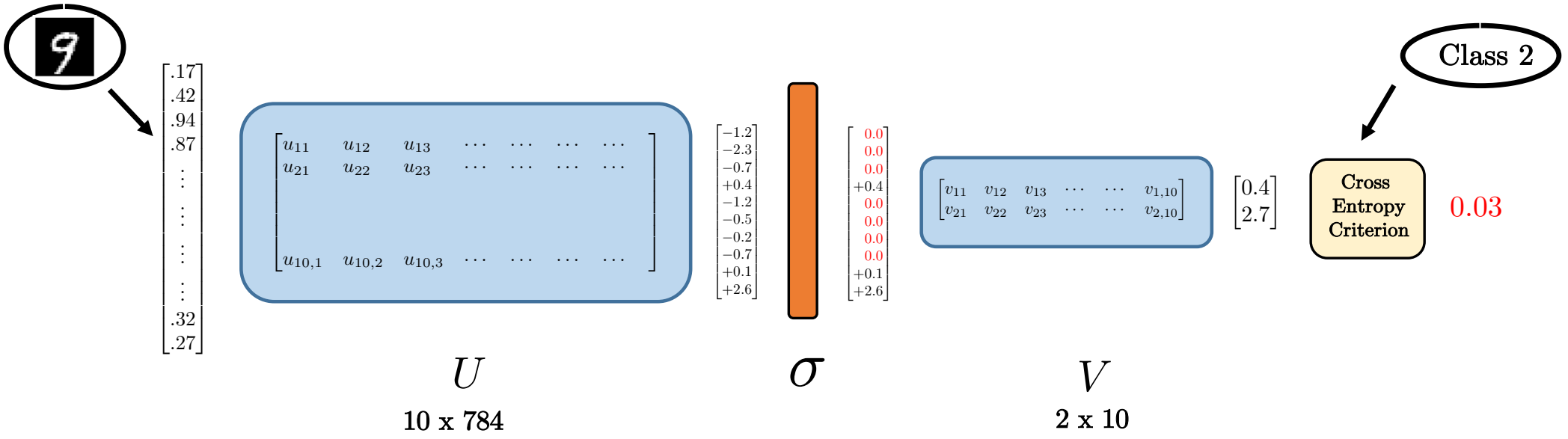


- It is hard to find good templates for small versus large numbers :



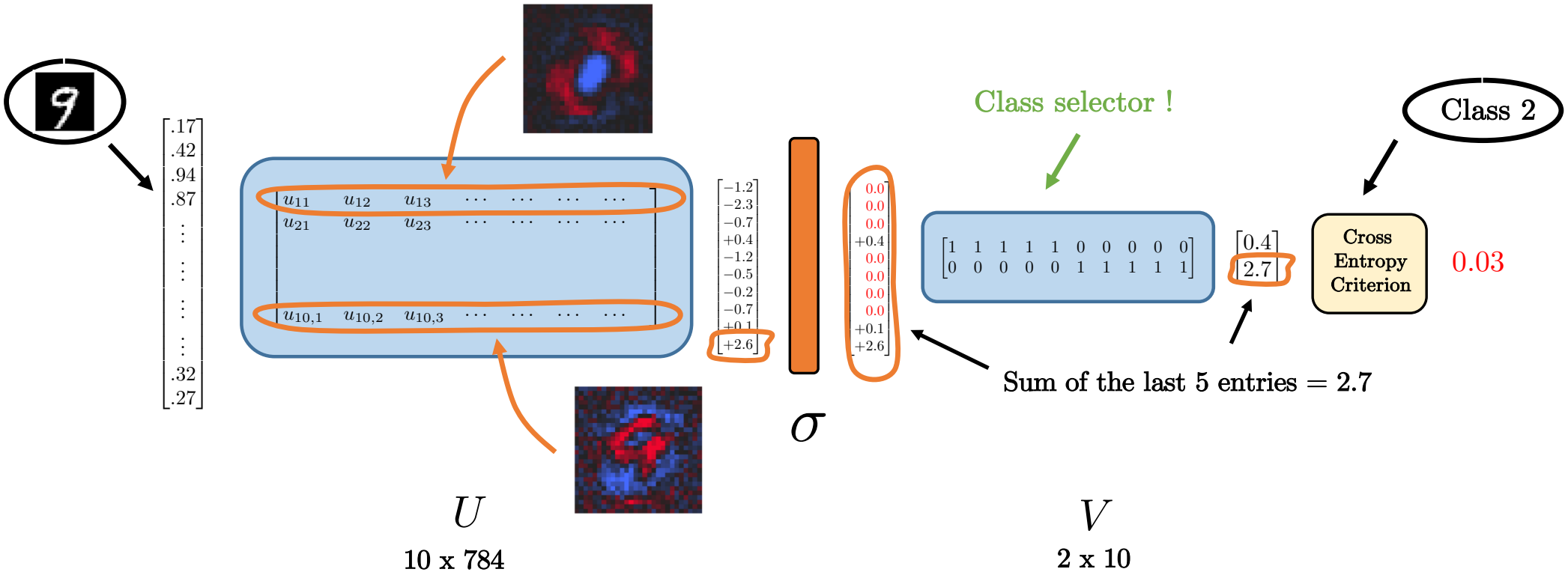
A hard classification problem

- Let's consider a two-layer network rather than just one-layer :



A hard classification problem

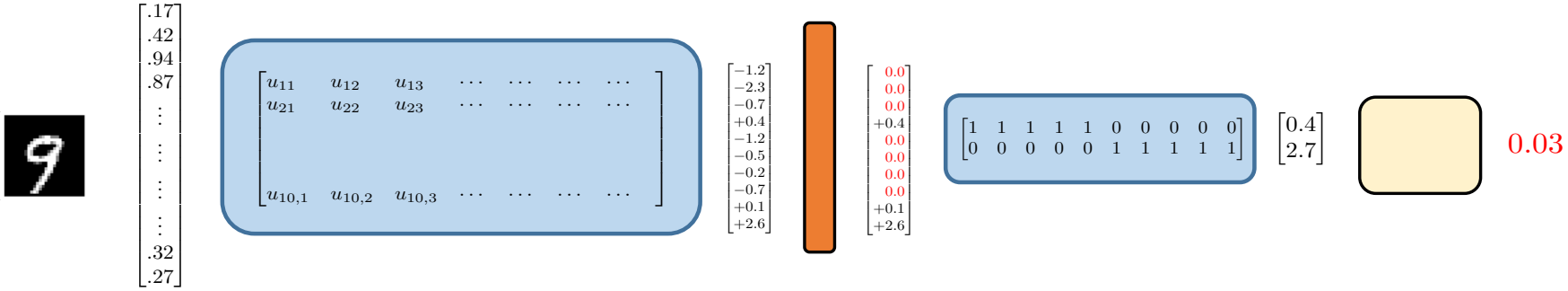
- After training, the network parameters U and V will become as follows :



This network will solve this classification problem.

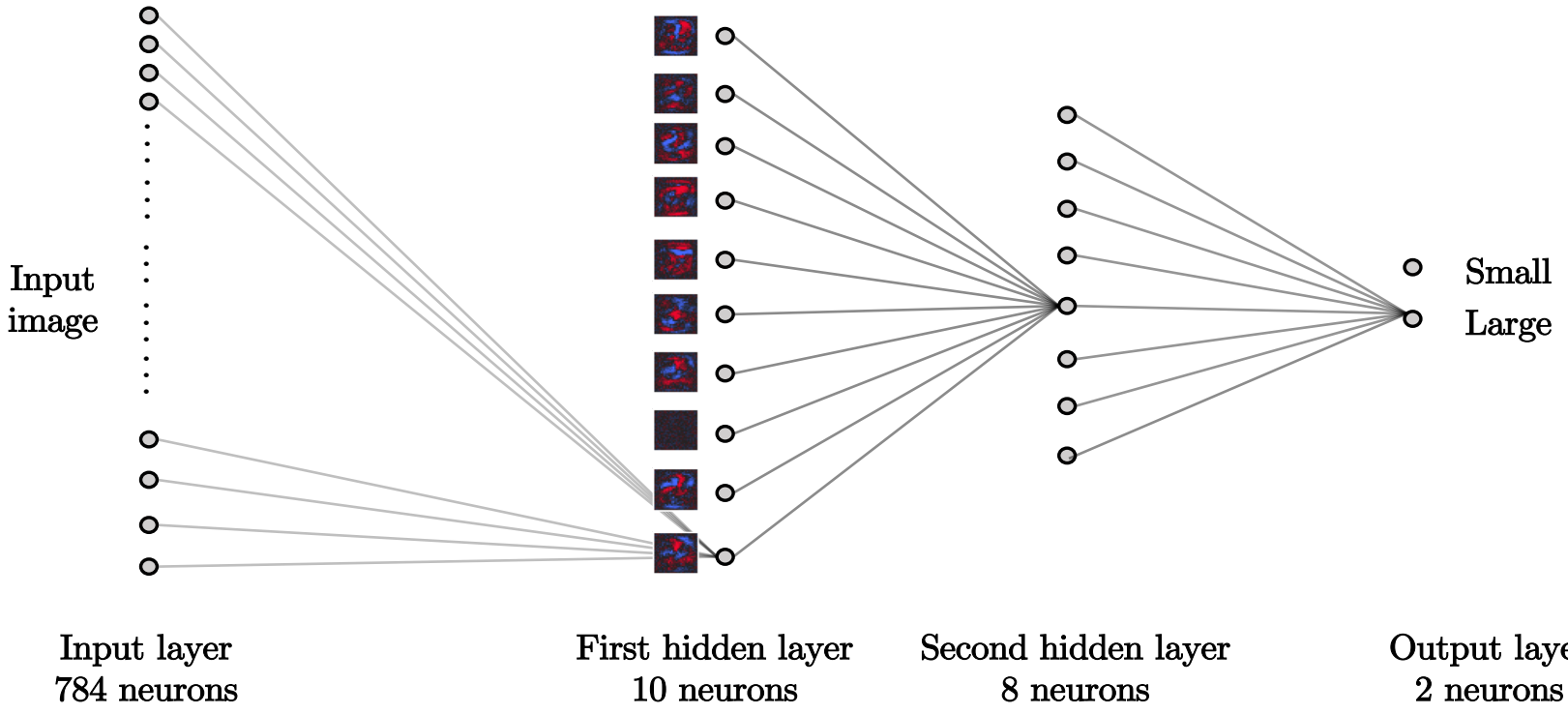
Multiple-step reasoning

- Multiple layers provide multiple-step reasoning :
  - The first layer is used to identify the type of digits
  - The second layer uses the classification made by the first layer, and processes it by pooling together 0-4 into one group, and 5-9 into another group.



Multiple-step reasoning

- What would happen with more layers?
  - The new network would also succeed.
  - The network would learn relevant features and do multiple stage reasoning.
  - The new features will be less/no interpretable (it is fine).

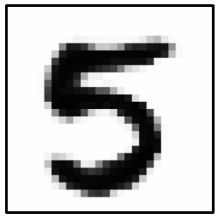


Outline

- The three representations of neural networks
- The importance of non-linearities
- The importance of multiple layers
- **Labs with MNIST and CIFAR**

MNIST dataset

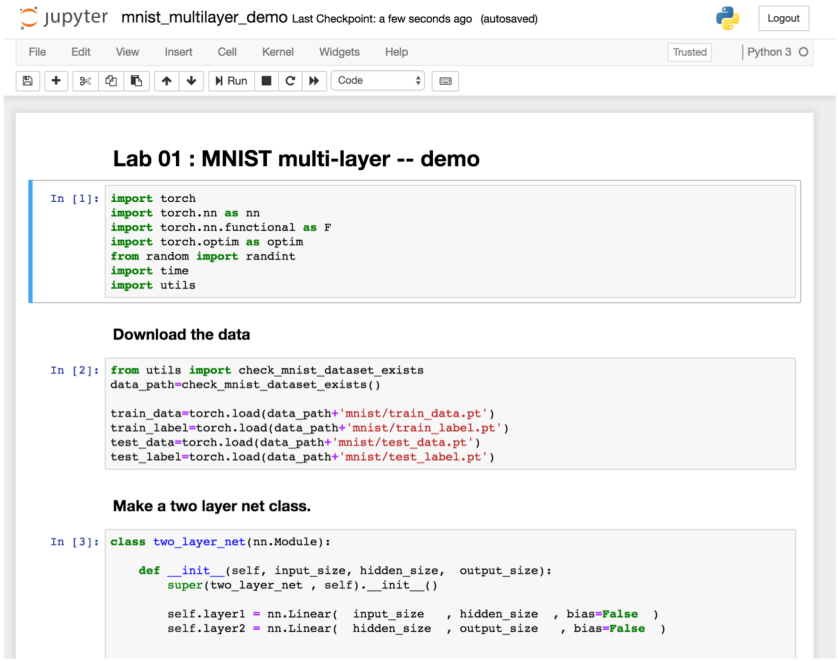
- Number of classes: 10
- Training set: 50,000 labelled images
- Test set: 10,000 labelled images



28 x 28 grayscale image

Lab 01

- MNIST classification with MLP



jupyter mnist_multilayer_demo Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 01 : MNIST multi-layer -- demo

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from random import randint
import time
import utils
```

Download the data

In [2]:
```
from utils import check_mnist_dataset_exists
data_path=check_mnist_dataset_exists()

train_data=torch.load(data_path+'mnist/train_data.pt')
train_label=torch.load(data_path+'mnist/train_label.pt')
test_data=torch.load(data_path+'mnist/test_data.pt')
test_label=torch.load(data_path+'mnist/test_label.pt')
```

Make a two layer net class.

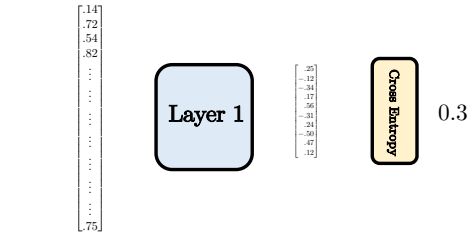
In [3]:
```
class two_layer_net(nn.Module):

    def __init__(self, input_size, hidden_size,  output_size):
        super(two_layer_net , self).__init__()

        self.layer1 = nn.Linear(  input_size   , hidden_size  , bias=False  )
        self.layer2 = nn.Linear(  hidden_size  , output_size   , bias=False  )
```

Lab 01

- Results
  - One-layer network:

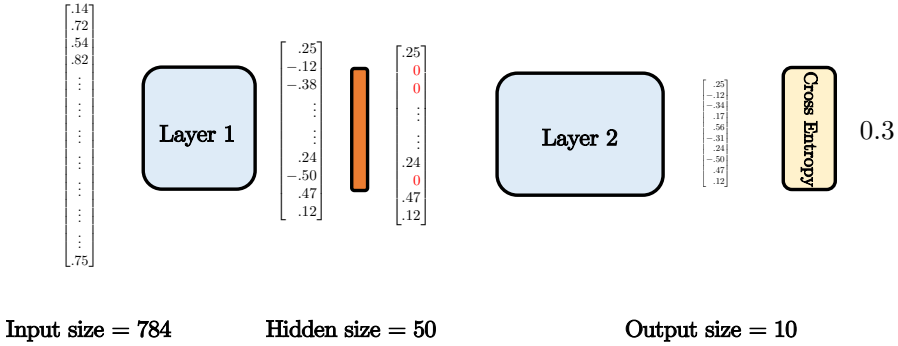


Number of parameters:
784 x 10 = 7,840

Error on train set: 8%

Error on test set: 8%

  - Two-layer network:



Number of parameters:
784 x 50 + 50 x 10 = 39,700

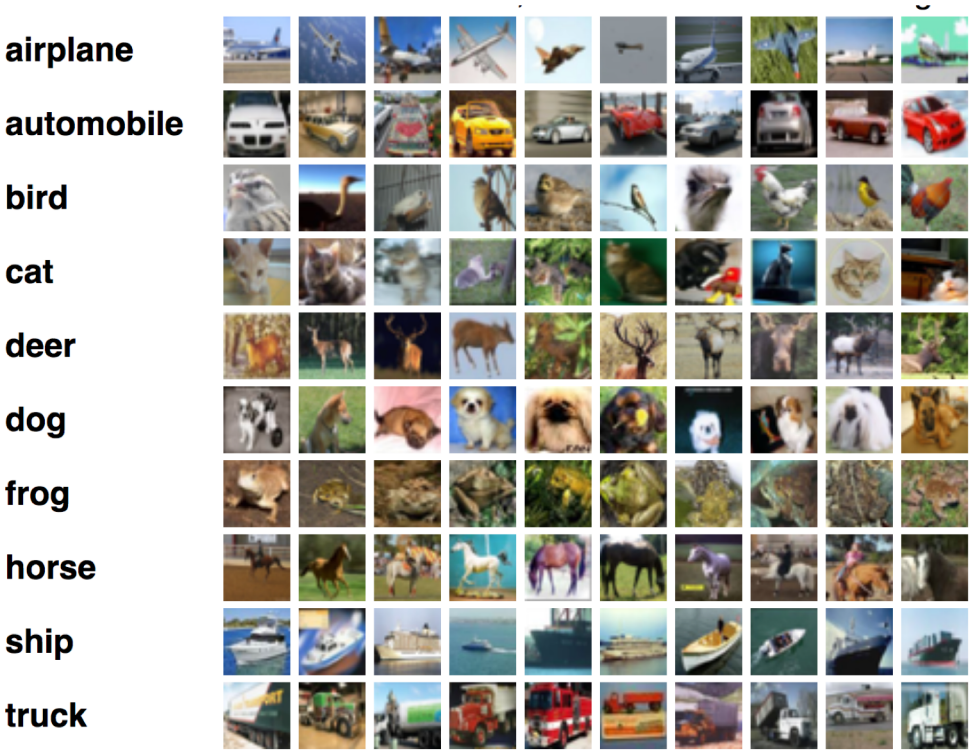
Error on train set: 0%

Error on test set: 2.5%

CIFAR dataset

- Number of classes: 10
- Training set: 50,000 labelled images
- Test set: 10,000 labelled images

32 x 32 RGB image



Lab 02

- CIFAR classification with MLP


jupyter gpu_demo Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 02 : GPU demo

```
In [1]: import torch
        import torch.nn as nn
        import torch.nn.functional as F
```

Set the device variable to be either cuda or cpu

```
In [9]: device = torch.device("cuda")

        #device = torch.device("cpu")

        print(device)
```

```
cpu
```

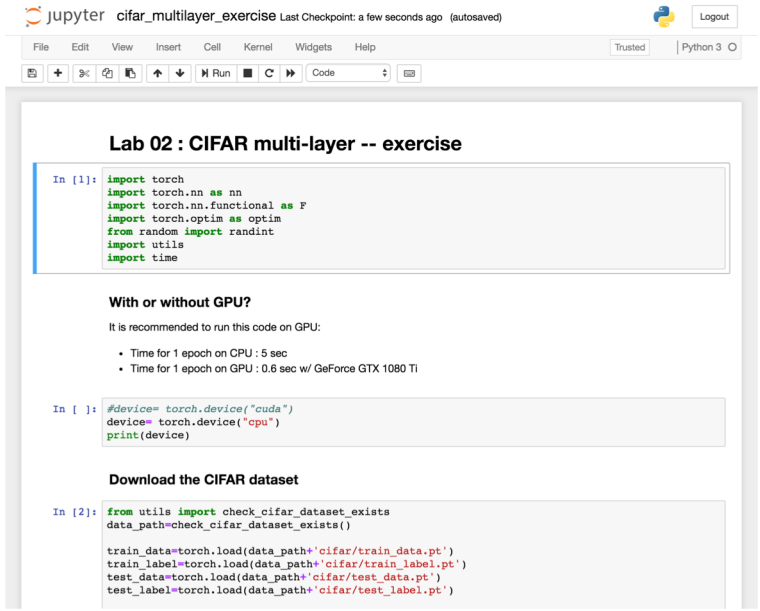
Make a random tensor (by default the tensor is created on the CPU)

```
In [10]: x=torch.rand(3,3)

         print(x)
```

```
tensor([[0.2126, 0.8232, 0.9466],
        [0.0370, 0.5176, 0.7032],
        [0.4544, 0.4457, 0.8515]])
```

Send the tensor to the device (which is going to be be the GPU if the device was set to be the GPU -- otherwise it will remain on the CPU)


jupyter cifar_multilayer_exercise Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 02 : CIFAR multi-layer -- exercise

```
In [1]: import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim
        from random import randint
        import utils
        import time
```

With or without GPU?

It is recommended to run this code on GPU:

- Time for 1 epoch on CPU : 5 sec
- Time for 1 epoch on GPU : 0.6 sec w/ GeForce GTX 1080 Ti

```
In [ ]: #device= torch.device("cuda")
        device= torch.device("cpu")
        print(device)
```

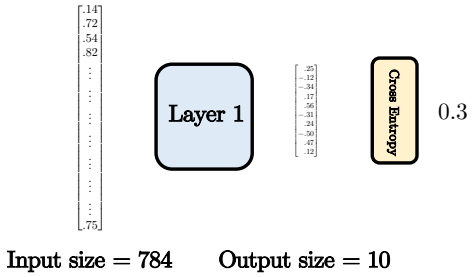
Download the CIFAR dataset

```
In [2]: from utils import check_cifar_dataset_exists
        data_path=check_cifar_dataset_exists()

        train_data=torch.load(data_path+'cifar/train_data.pt')
        train_label=torch.load(data_path+'cifar/train_label.pt')
        test_data=torch.load(data_path+'cifar/test_data.pt')
        test_label=torch.load(data_path+'cifar/test_label.pt')
```

Lab 02

- Results
  - One-layer network:

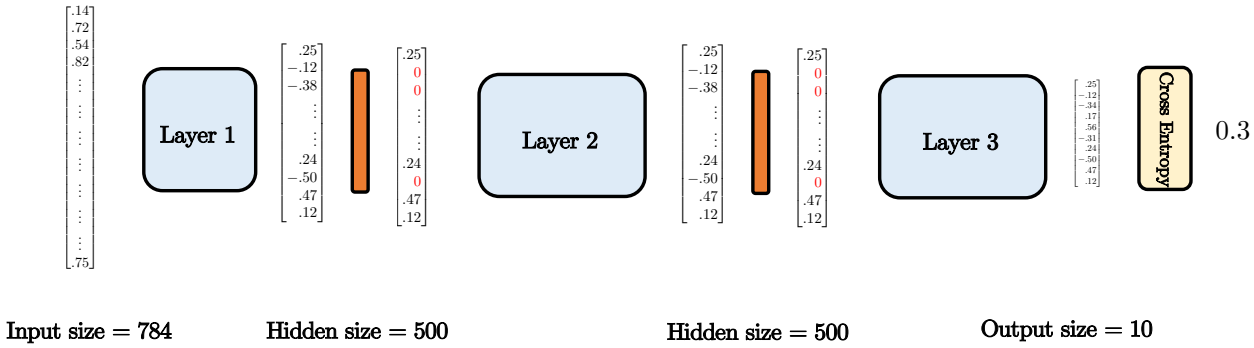


Number of parameters:
3,072 x 10 = 30,720

Error on train set: 59%

Error on test set: 65%

  - Three-layer network:



Number of parameters:
2 millions

Error on train set: 0%

Error on test set: 46%

Questions?