Neural Network Basics

TUYEN NGOC LE

Outline

1. Artificial Neural Networks and their relation to biology

2. The seminal Perceptron algorithm

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

4. How to train neural networks using the Keras library

Types of Neural Networks

There are many types of neural networks available or that might be in the development stage. They can be classified depending on their: Structure, Data flow, Neurons used and their density, Layers and their depth activation filters etc.

A. Perceptron

B. Feed Forward Neural Networks

C. Multilayer Perceptron

D. Convolutional Neural Network

E. Radial Basis Function Neural Networks

F. Recurrent Neural Networks

G. Sequence to sequence models

H. Modular Neural Network

Outline

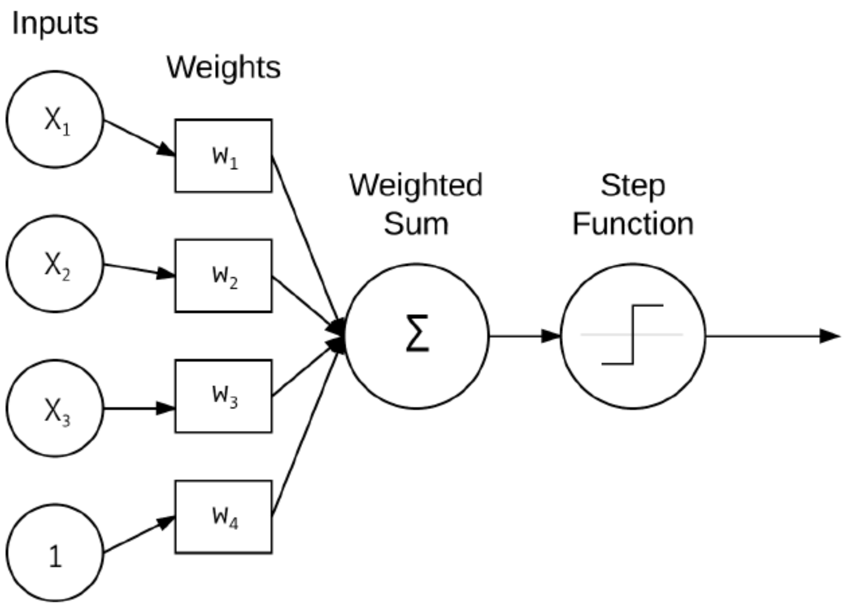
1. Artificial Neural Networks and their relation to biology

2. The seminal Perceptron algorithm

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

4. How to train neural networks using the Keras library

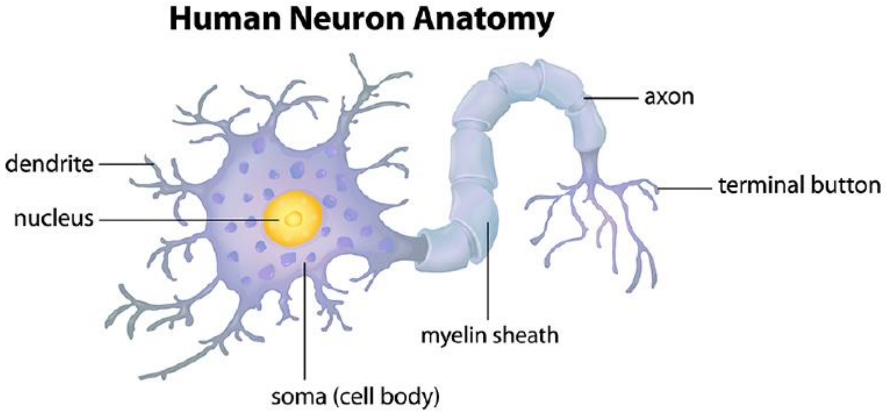
1. Artificial Neural Networks and their relation to biology



A simple NN that takes the weighted sum of the input *x* and weights *w*. This weighted sum is then passed through the activation function to determine if the neuron fires

- $f(w_1x_1 + w_2x_2 + \cdots + w_nx_n)$
- $f(\sum_{i=1}^{n} w_ix_i)$
- Or simply, $f(net)$, where $net = \sum_{i=1}^{n} w_ix_i$

Relation to Biology



The structure of a biological neuron. Neurons are connected to other neurons through their dendrites and enurons.

Activation Functions

$$f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

1. Artificial Neural Networks and their relation to biology

Activation Functions

1. step: $f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$

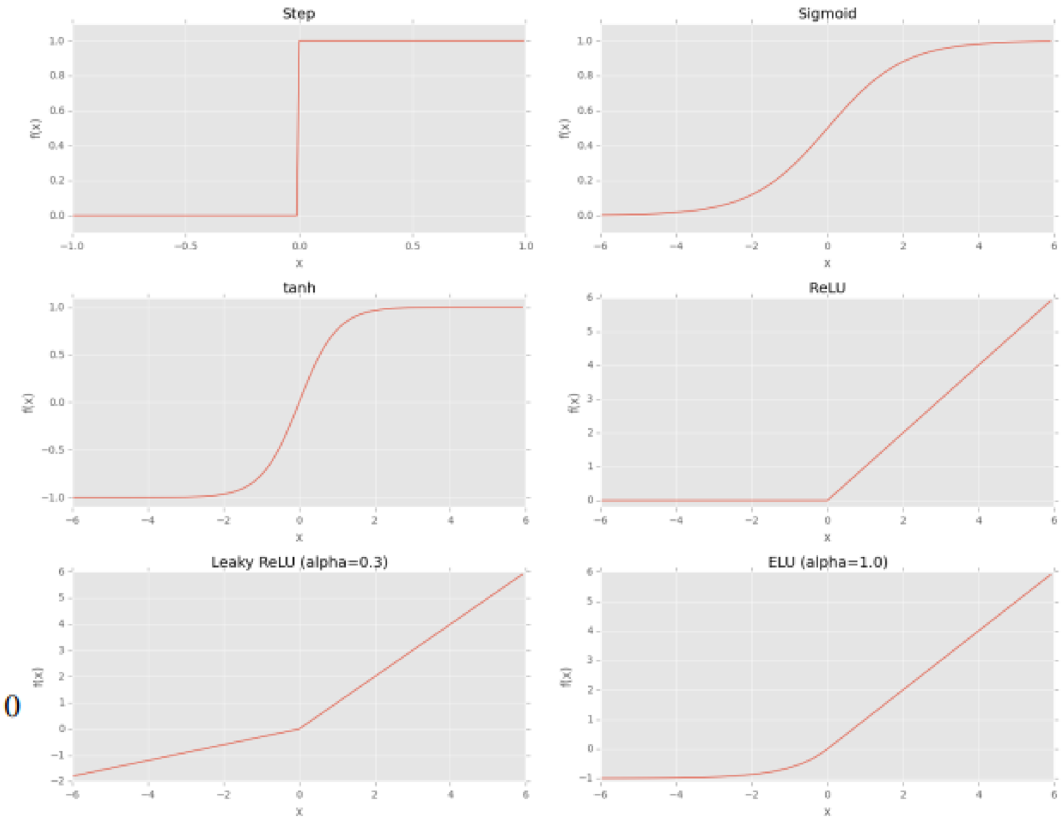
2. sigmoid: $t = \sum_{i=1}^{n} w_i x_i$ $\quad s(t) = 1/(1+e^{-t})$

3. tanh: $f(z) = tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$

4. Rectified Linear Unit (ReLU) $f(x) = max(0, x)$

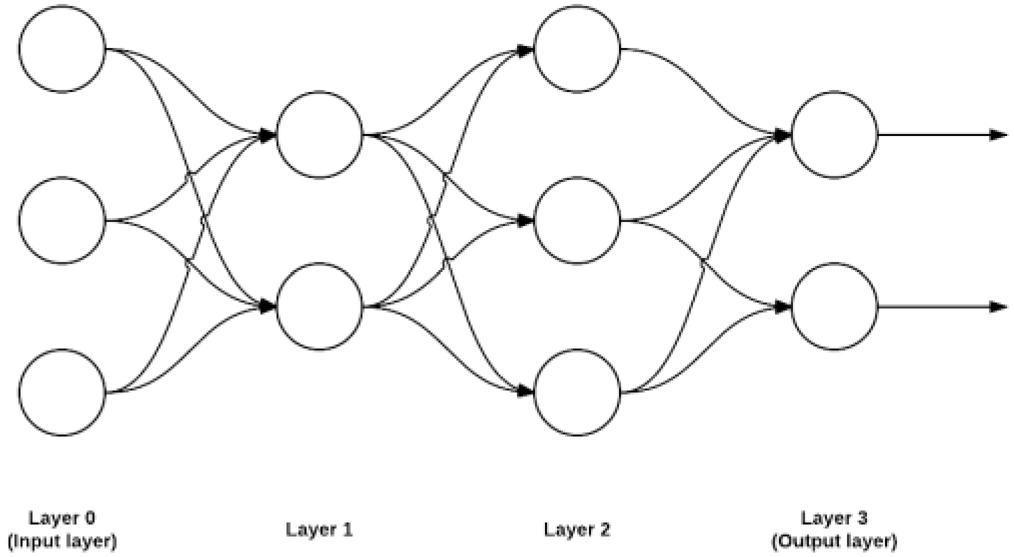
5. Leaky ReLUs $f(net) = \begin{cases} net & \text{if } net >= 0 \\ \alpha \times net & \text{otherwise} \end{cases}$

6. Exponential Linear Units (ELUs) $f(net) = \begin{cases} net & \text{if } net >= 0 \\ \alpha \times (exp(net) - 1) & \text{otherwise} \end{cases}$

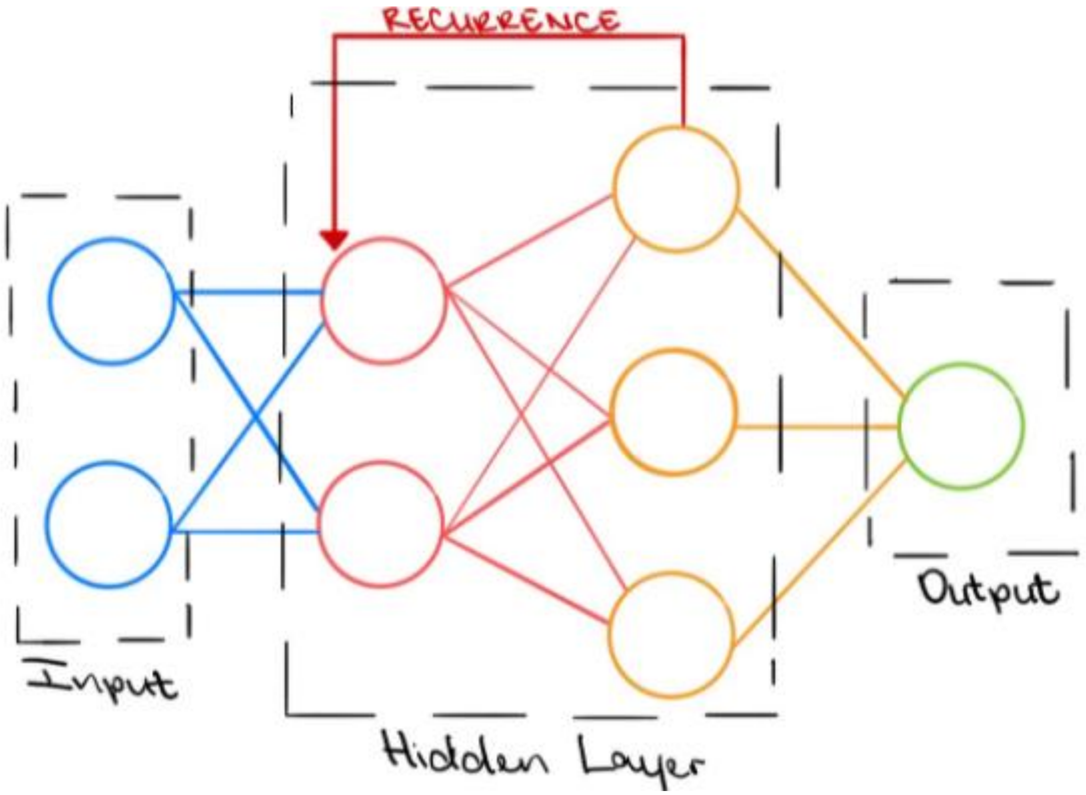


1. Artificial Neural Networks and their relation to biology

Feedforward Network Architectures



Recurrent neural networks



Outline

1. Artificial Neural Networks and their relation to biology

2. The seminal Perceptron algorithm

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

4. How to train neural networks using the Keras library

2. The Perceptron algorithm

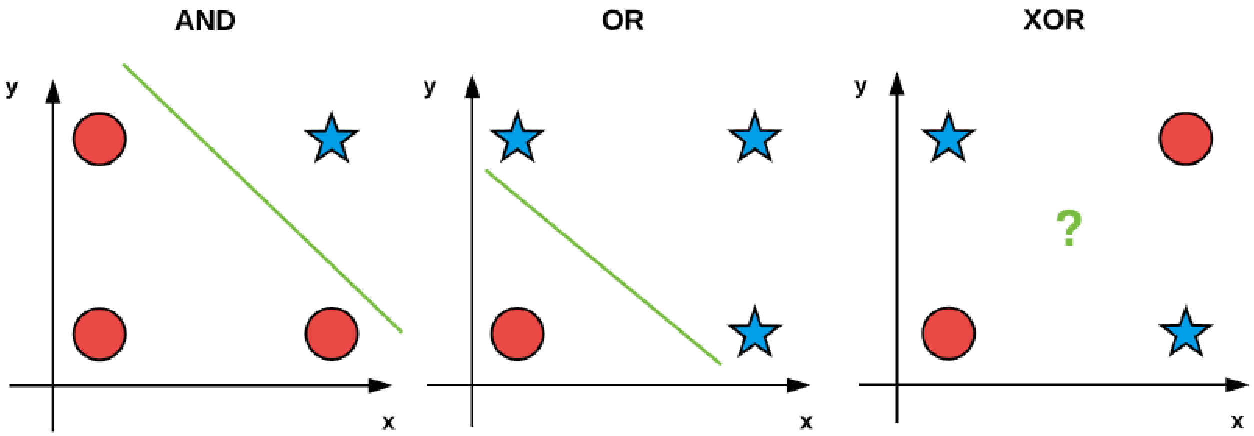
AND, OR, and XOR Datasets

x_0	x_1	$x_0 \& x_1$
0	0	0
0	1	0
1	0	0
1	1	1

x_0	x_1	$x_0 \vert x_1$
0	0	0
0	1	1
1	0	1
1	1	1

x_0	x_1	$x_0 \wedge x_1$
0	0	0
0	1	1
1	0	1
1	1	0

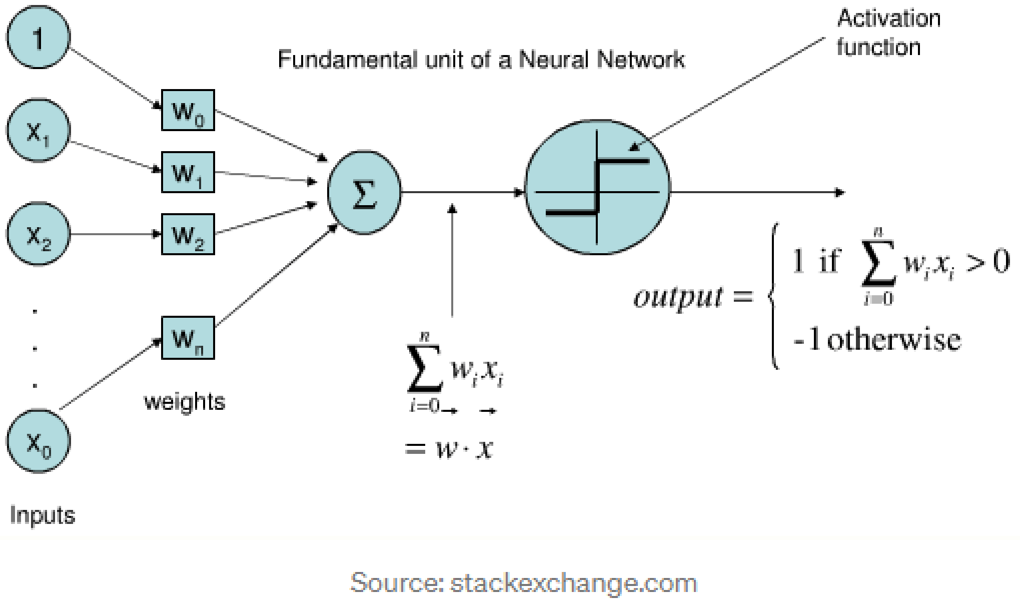
- The AND and OR bitwise datasets are linearly separable (we can draw a single line (green) that separates the two classes)
- For XOR bitwise dataset→it is impossible to draw a single line that separates the two classes (nonlinearly separable dataset).



- Perceptron algorithm can correctly classify the AND and OR functions but fails to classify the XOR data

2. The Perceptron algorithm

Training a Perceptron is a fairly straightforward operation. Our goal is to obtain a set of weights ***w*** that accurately classifies each instance in our training set. **In order to train our Perceptron, we iteratively feed the network our training data multiple times. Each time the network has seen the full set of training data, we say an epoch has passed**. It normally takes many epochs until a weight vector ***w*** can be learned to linearly separate our two classes of data.



Architecture of the Perceptron network

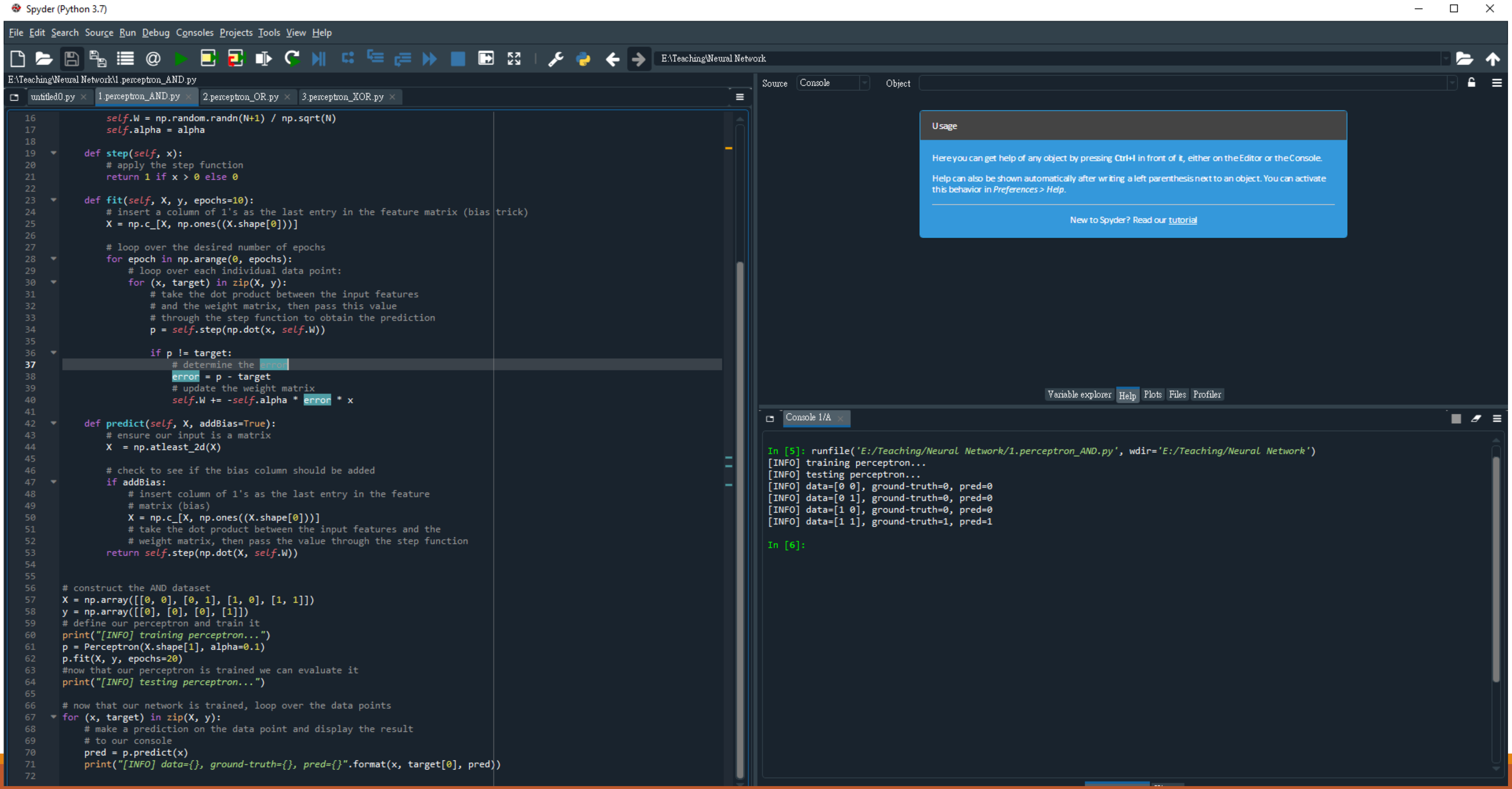
2. The Perceptron algorithm

Perceptron Training Procedure and the Delta Rule

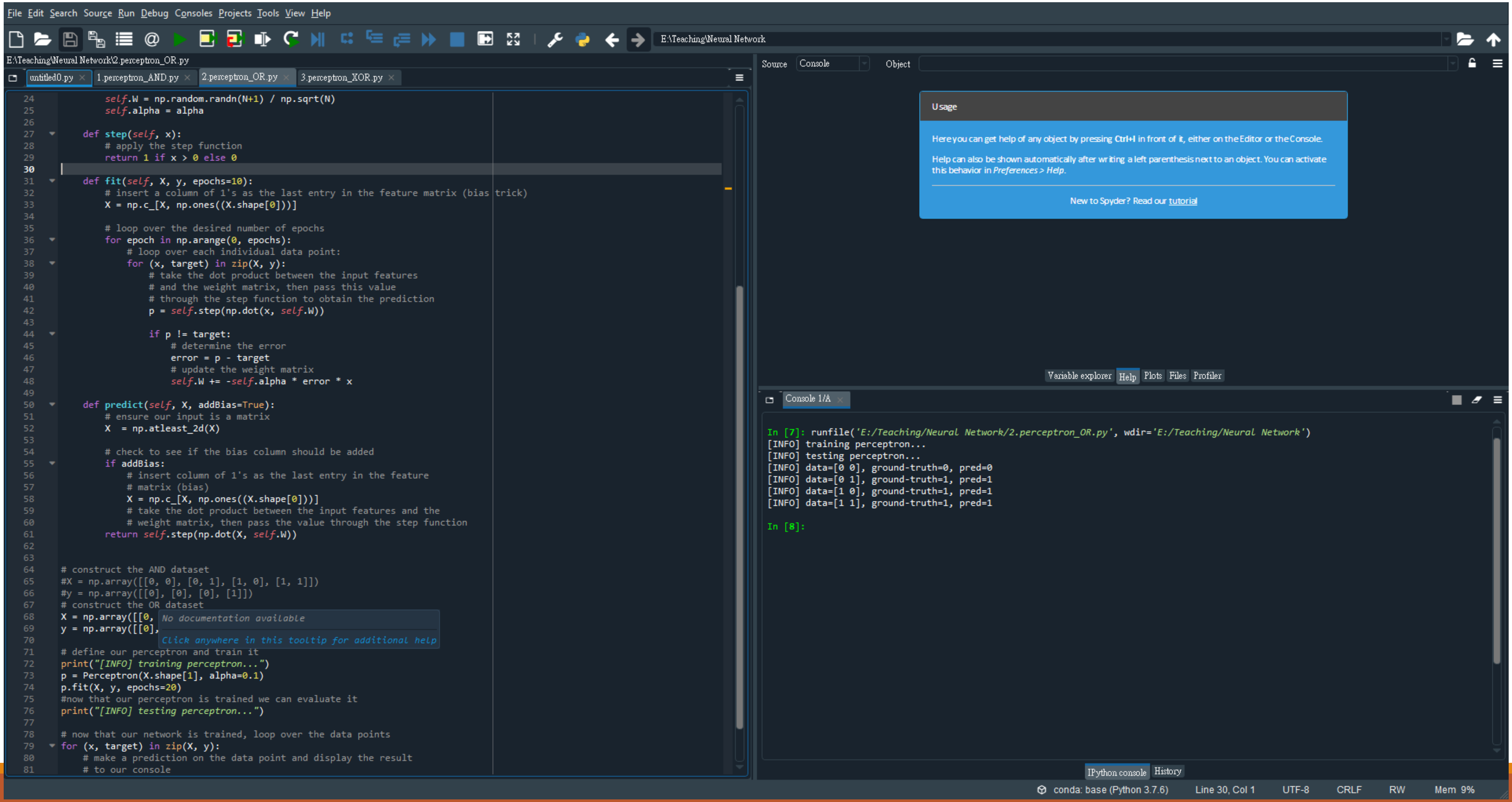
1. Initialize our weight vector $\boldsymbol{w}$ with small random values
2. Until Perceptron converges:
   (a) Loop over each feature vector x_j and true class label d_i in our training set D
   (b) Take $\boldsymbol{x}$ and pass it through the network, calculating the output value: $y_j = f(\boldsymbol{w}(\boldsymbol{t}) \cdot \boldsymbol{x}_j)$
   (c) Update the weights $\boldsymbol{w}$: $w_i(t+1) = w_i(t) + \eta(d_j - y_j)x_{j,i}$ for all features $0 <= i <= n$

η: learning rate

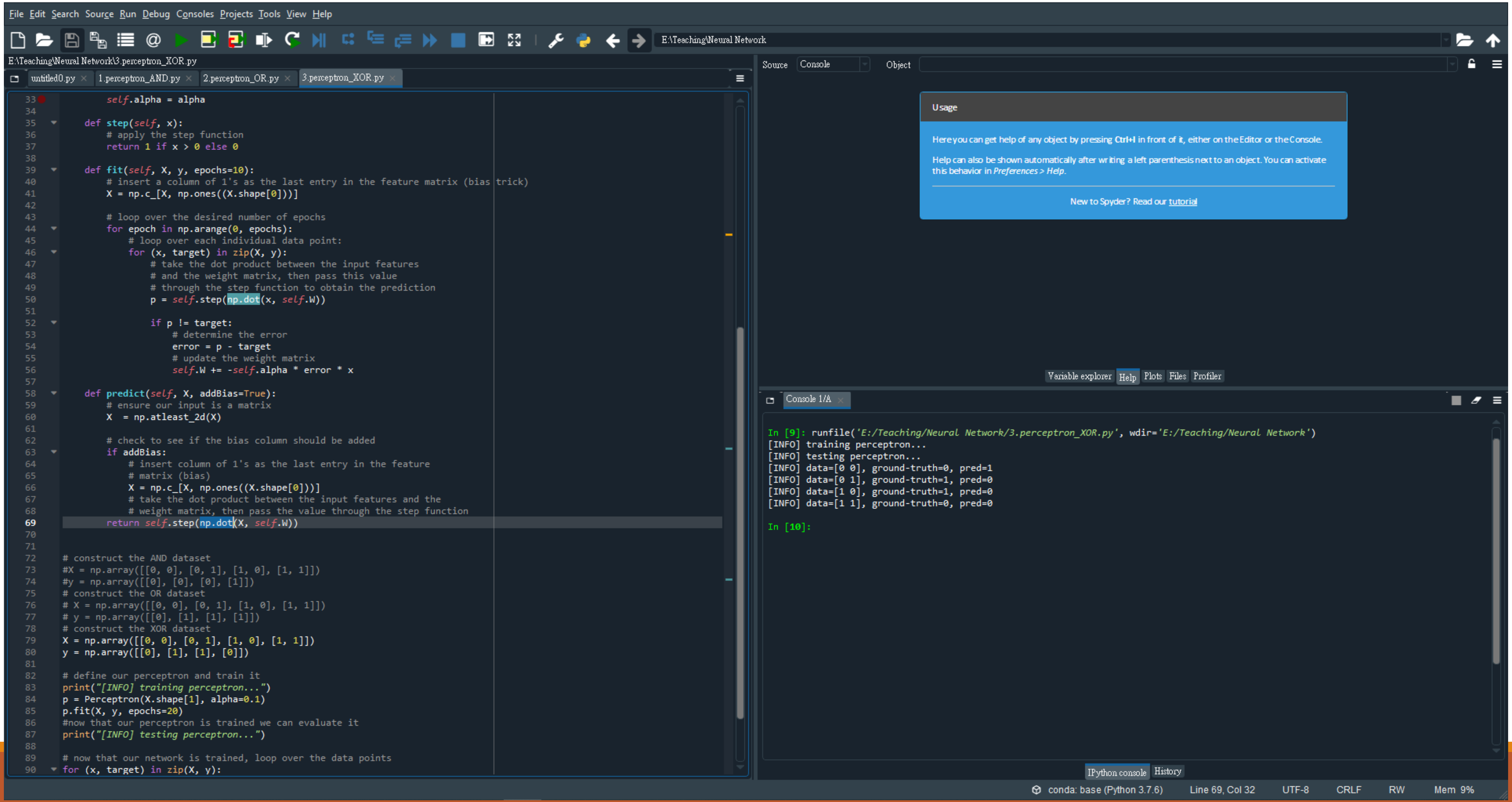
2. The Perceptron algorithm



2. The Perceptron algorithm



2. The Perceptron algorithm



2. The Perceptron algorithm

What is a function in Python?

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function

```
def function_name(parameters):
	"""docstring"""
	statement(s)
```

Above shown is a function definition that consists of the following components.

1. Keyword `def` that marks the start of the function header.
2. A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of the function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
7. An optional `return` statement to return a value from the function.

https://www.programiz.com/python-programming/function

What is the use of Self in Python?

The self is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python. It binds the attributes with the given arguments. The reason why we use self is that Python does not use the '@' syntax to refer to instance attributes. Join our Master Python programming course to know more. In Python, we have methods that make the instance to be passed automatically, but not received automatically.

Example:

```
class food():

# init method or constructor
def __init__(self, fruit, color):
self.fruit = fruit
self.color = color

def show(self):
print("fruit is", self.fruit)
print("color is", self.color )

apple = food("apple", "red")
grapes = food("grapes", "green")

apple.show()
grapes.show()
```

https://www.edureka.co/blog/self-in-python/

2. The Perceptron algorithm

Python zip()

In this tutorial, we will learn about the Python zip() function with the help of examples.

The `zip()` function takes iterables (can be zero or more), aggregates them in a tuple, and returns it.

Example

```
languages = ['Java', 'Python', 'JavaScript']
versions = [14, 3, 6]

result = zip(languages, versions)
print(list(result))

# Output: [('Java', 14), ('Python', 3), ('JavaScript', 6)]
```

Syntax of zip()

The syntax of the `zip()` function is:

```
zip(*iterables)
```

https://www.programiz.com/python-programming/methods/built-in/zip

numpy.dot¶

`numpy.dot(a, b, out=None)`

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using `matmul` or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to `multiply` and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where `M>=2`), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

Parameters:
a : *array_like*
First argument.
b : *array_like*
Second argument.
out : *ndarray, optional*
Output argument. This must have the exact kind that would be returned if it was not used. In particular, it must have the right type, must be C-contiguous, and its dtype must be the dtype that would be returned for *dot(a,b)*. This is a performance feature. Therefore, if these conditions are not met, an exception is raised, instead of attempting to be flexible.

Returns:
output : *ndarray*
Returns the dot product of *a* and *b*. If *a* and *b* are both scalars or both 1-D arrays then a scalar is returned; otherwise an array is returned. If *out* is given, then it is returned.

Raises:
ValueError
If the last dimension of *a* is not the same size as the second-to-last dimension of *b*.

https://numpy.org/doc/stable/reference/generated/numpy.dot.html

Outline

1. Artificial Neural Networks and their relation to biology

2. The seminal Perceptron algorithm

3. **The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently**

4. How to train neural networks using the Keras library

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

Backpropagation can be considered the cornerstone of modern neural networks and deep learning.

The backpropagation algorithm consists of two phases:

1. The forward pass where our inputs are passed through the network and output predictions obtained (also known as the propagation phase).

2. The backward pass where we compute the gradient of the loss function at the final layer (i.e., predictions layer) of the network and use this gradient to recursively apply the chain rule to update the weights in our network (also known as the weight update phase).

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

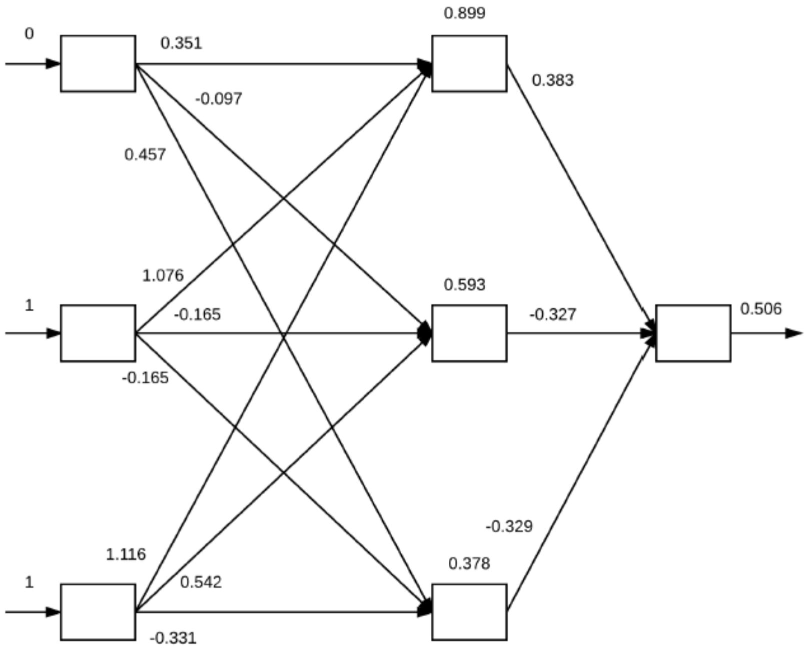
The Forward Pass

Let's compute the inputs to the three nodes in the hidden layers:

1. $\sigma((0 \times 0.351) + (1 \times 1.076) + (1 \times 1.116)) = 0.899$
2. $\sigma((0 \times -0.097) + (1 \times -0.165) + (1 \times 0.542)) = 0.593$
3. $\sigma((0 \times 0.457) + (1 \times -0.165) + (1 \times -0.331)) = 0.378$

$$\sigma((0.899 \times 0.383) + (0.593 \times -0.327) + (0.378 \times -0.329)) = 0.506$$

$$f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$



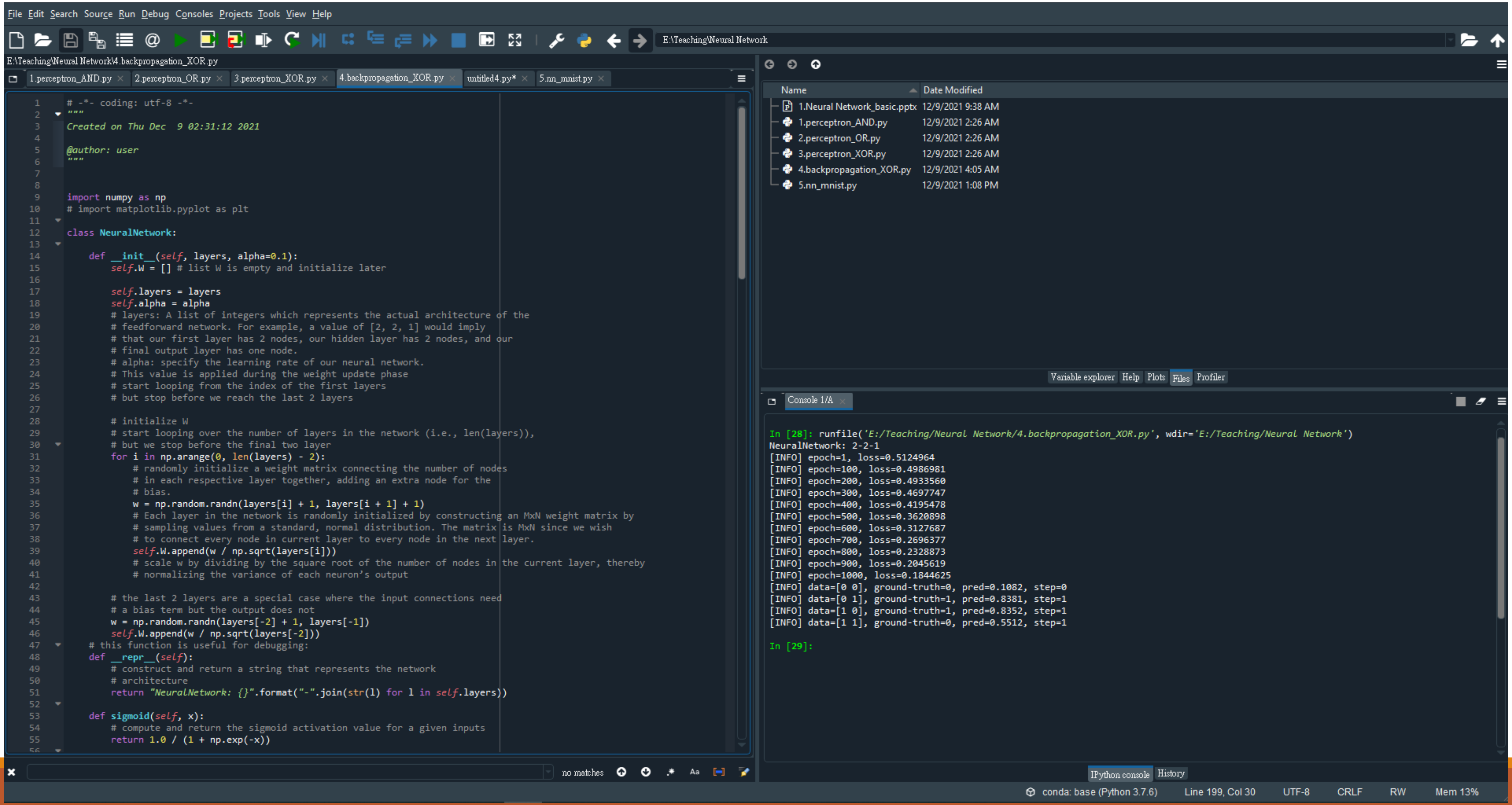
An example of the forward propagation pass. The input vector [0,1,1] is presented to the network. The dot product between the inputs and weights are taken, followed by applying the sigmoid activation function to obtain the values in the hidden layer (0.899, 0.593, and 0.378, respectively). Finally, the dot product and sigmoid activation function is computed for the final layer, yielding an output of 0.506. Applying the step function to 0.506 yields 1, which is indeed the correct target class label.

The Backward Pass

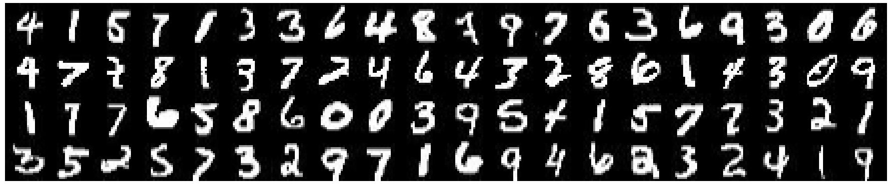
In order to apply the backpropagation algorithm, our activation function must be ***differentiable*** so that we can compute the ***partial derivative*** of the error with respect to a given weight w_{0j}, *loss*(E), node output o_j, and network output net_j.

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{i,j}}$$

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

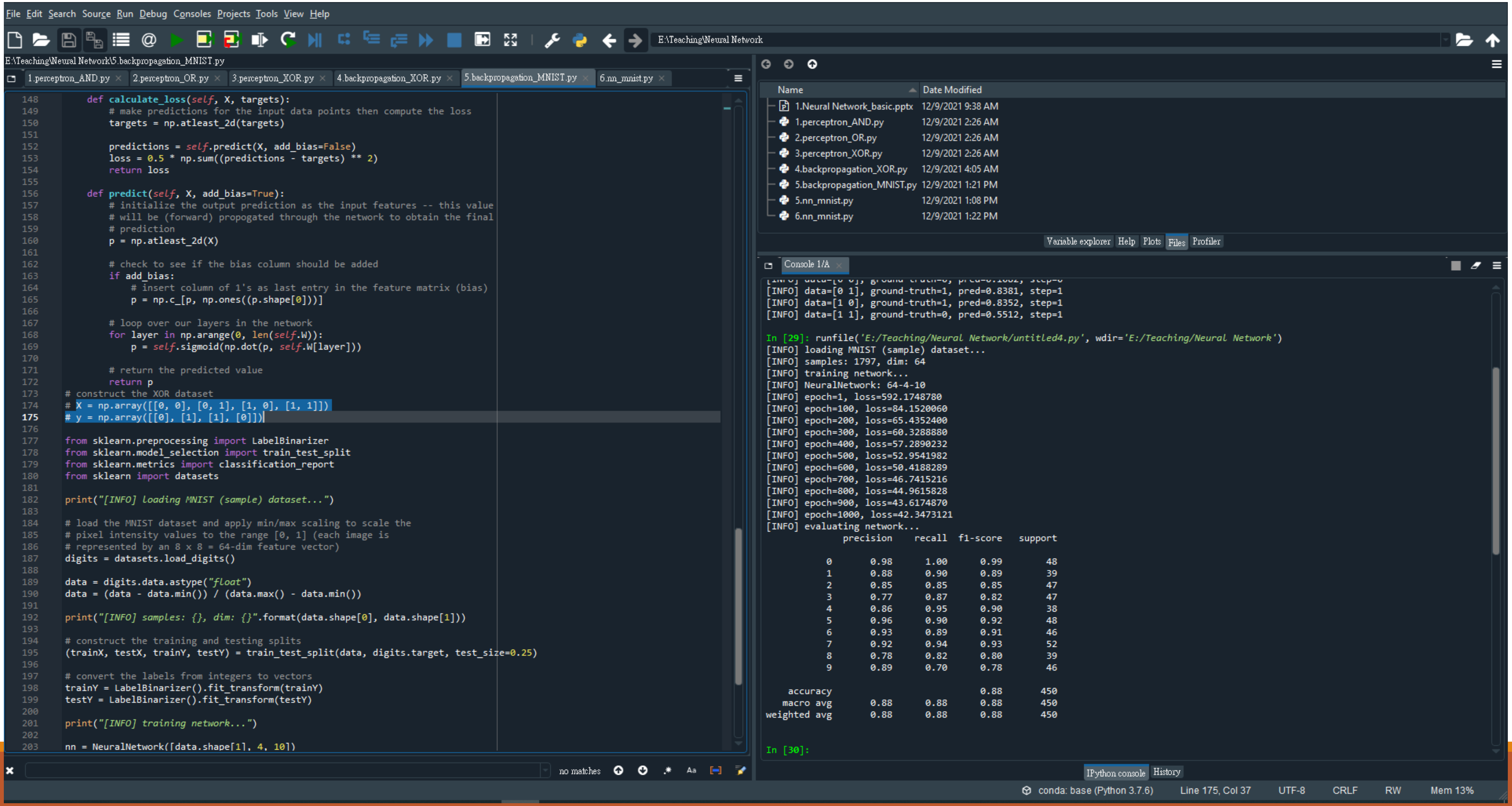


3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently



MNIST dataset: handwritten digit recognition. This subset of the MNIST dataset is built-into the scikit-learn library and includes 1,797 example digits, each of which are 88 grayscale images (the original images are 2828. When flattened, these images are represented by an 88 = 64-dim vector.

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently



Outline

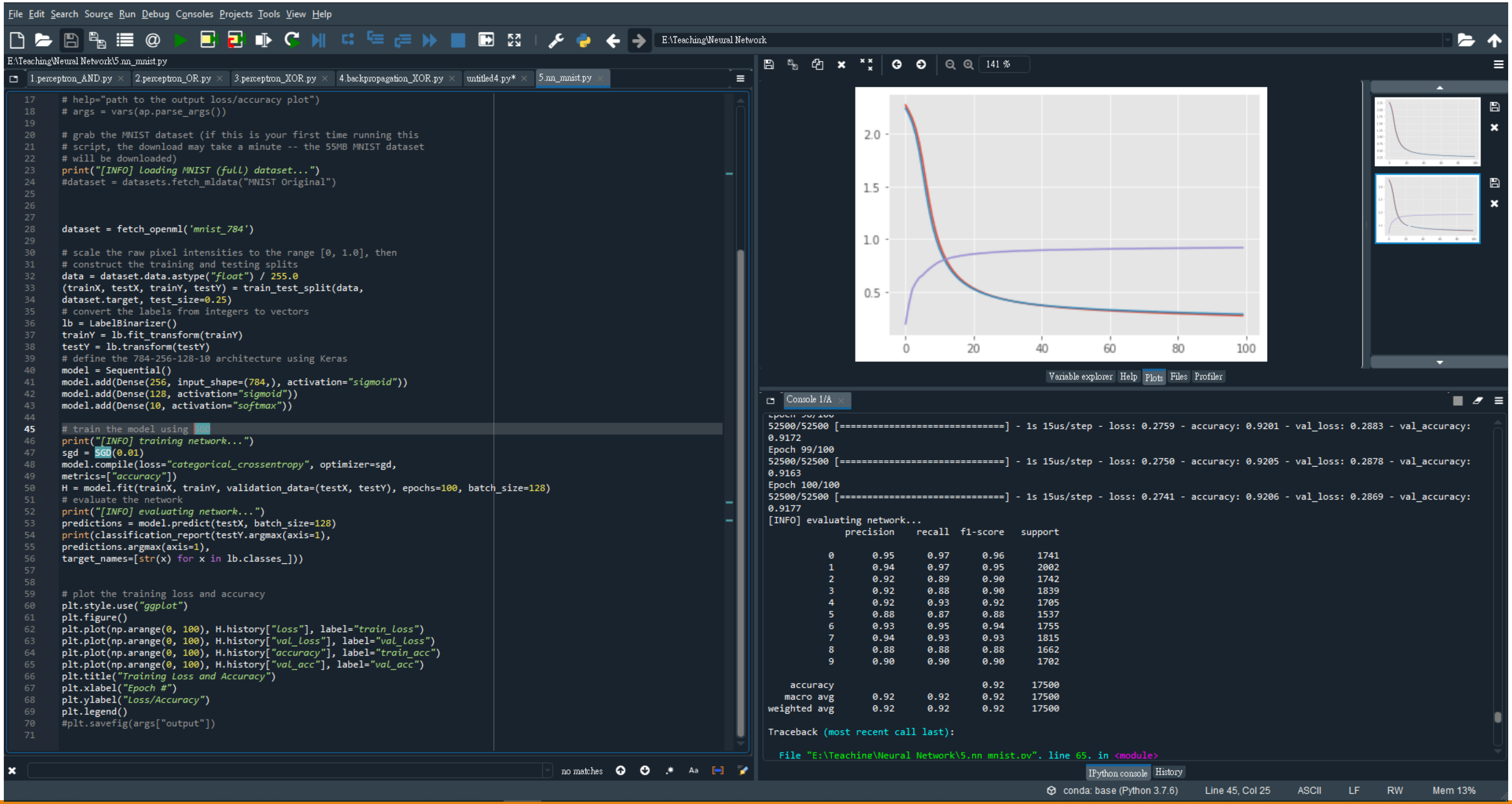
1. Artificial Neural Networks and their relation to biology

2. The seminal Perceptron algorithm

3. The backpropagation algorithm and how it can be used to train multi-layer neural networks efficiently

4. How to train neural networks using the Keras library

4. How to train neural networks using the Keras library



See you next week