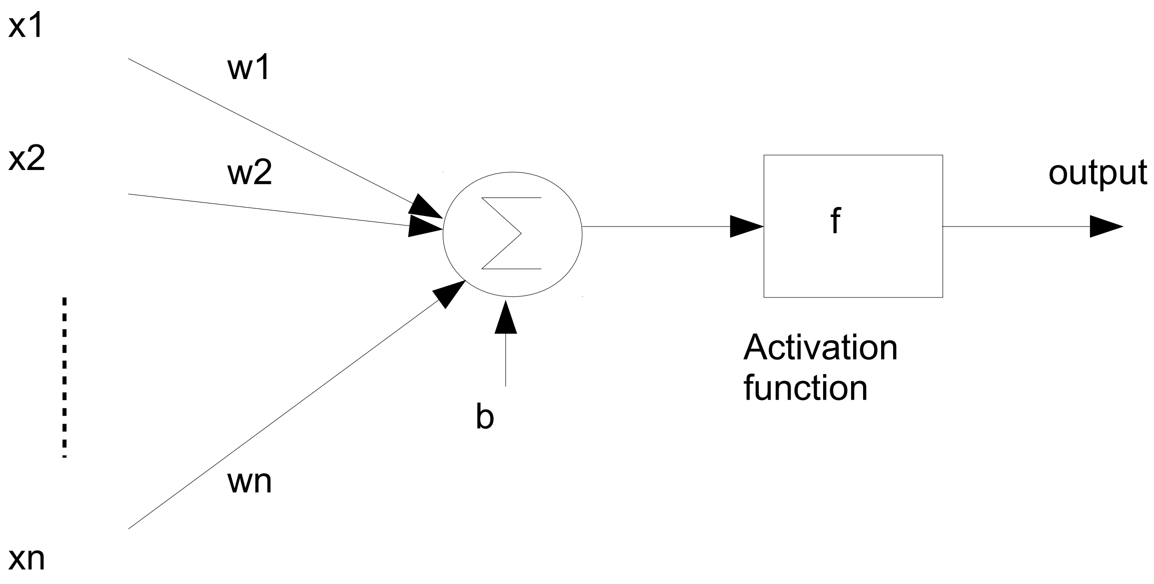
Intro-python-for-nn-2019-2-28.odp

Harry Li, Ph.D.

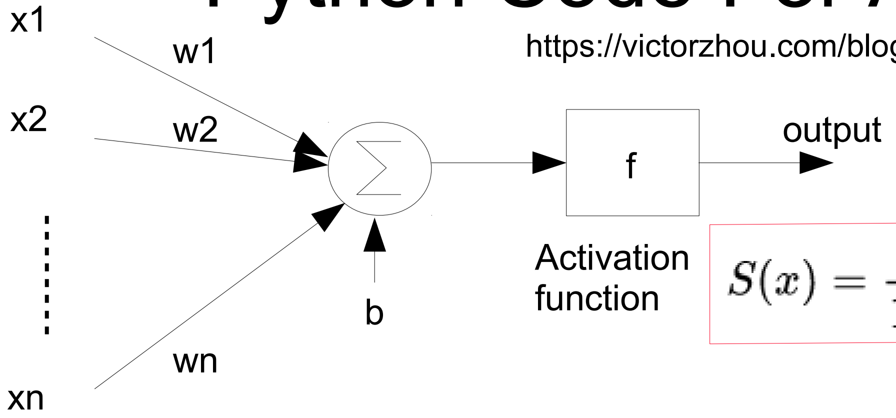
Neural Network Basic Building Block



Harry Li, Ph.D.

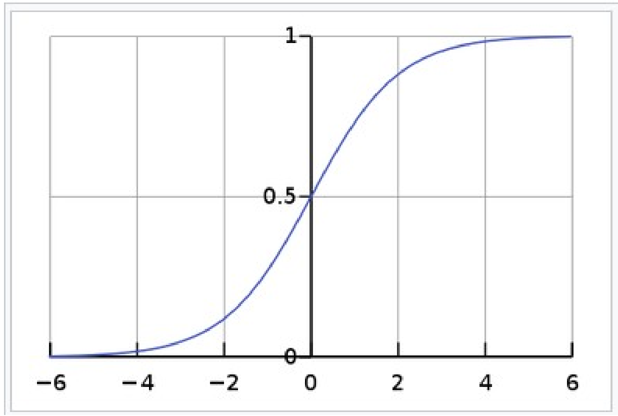
Python Code For A Single Neuron

https://victorzhou.com/blog/intro-to-neural-networks/



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

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve



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

Note: 1. define activation function

```
def sigmoid(x):
  # Our activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))
```

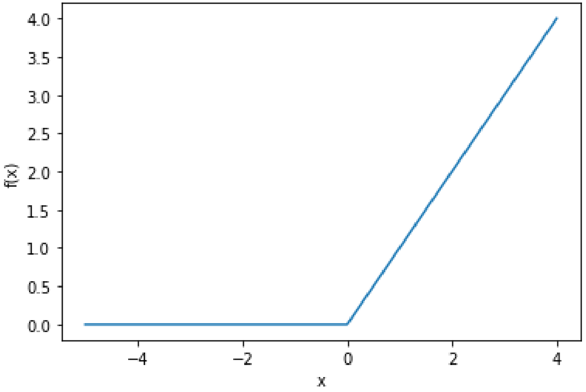
2. define a single neuron

```
class Neuron:
  def __init__(self, weights, bias):
    self.weights = weights
    self.bias = bias

  def feedforward(self, inputs):
    # Weight inputs, add bias, then use the activation function
    total = np.dot(self.weights, inputs) + self.bias
    return sigmoid(total)
```

Reference: for further discussion on Sigmoid
https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function

Reference: RELU activation



Harry Li, Ph.D.

Python Classes Example

https://www.tutorialspoint.com/python3/python_classes_objects.htm

```
Create a python class with keyword class, then the name of
the class, then : sign
```

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

1. The class has a documentation string, which can be accessed via ClassName.__doc__.

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)
   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ", self.salary)
```

2. class variable whose value is shared among all the instances of a in this class.

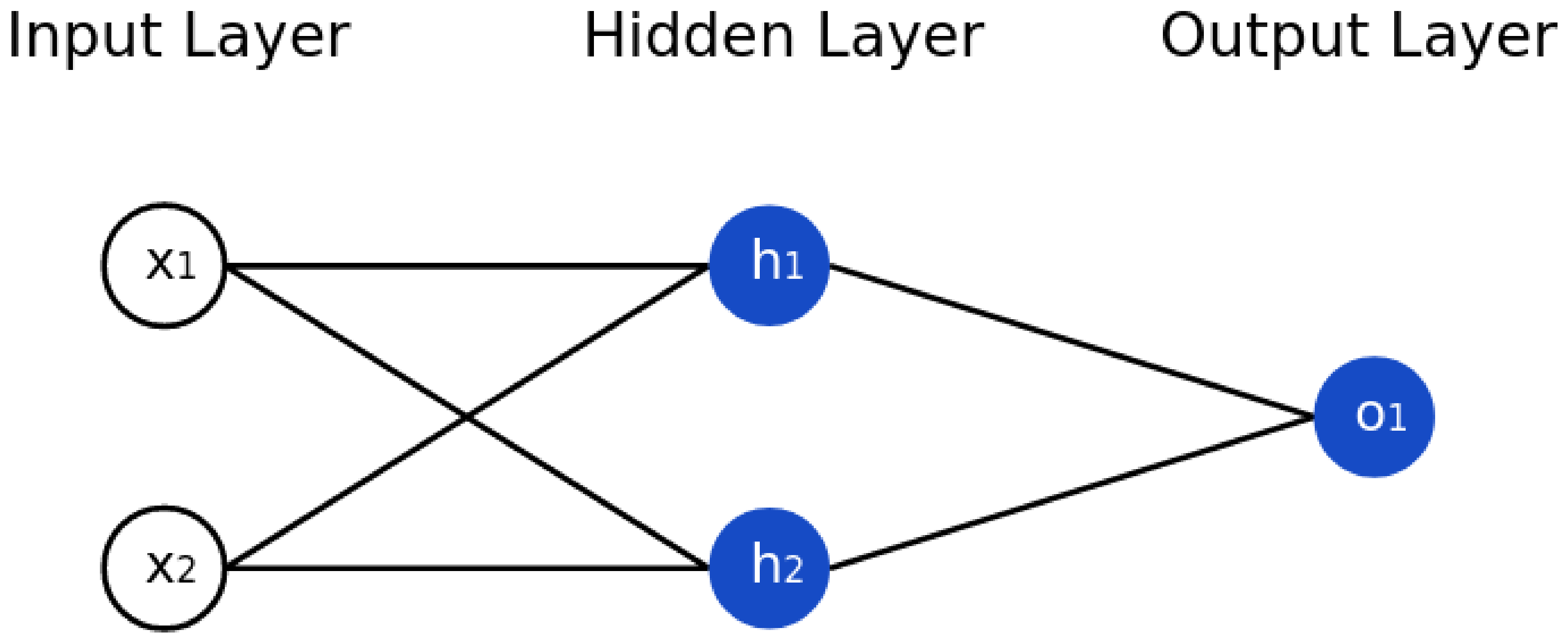
3. The first method __init__() is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.

4. You declare other class methods like normal functions with the exception that the first argument to each method is self. Python adds the self argument to the list for you; you do not need to include it when you call the methods.

Harry Li, Ph.D.

Feed Forward NN

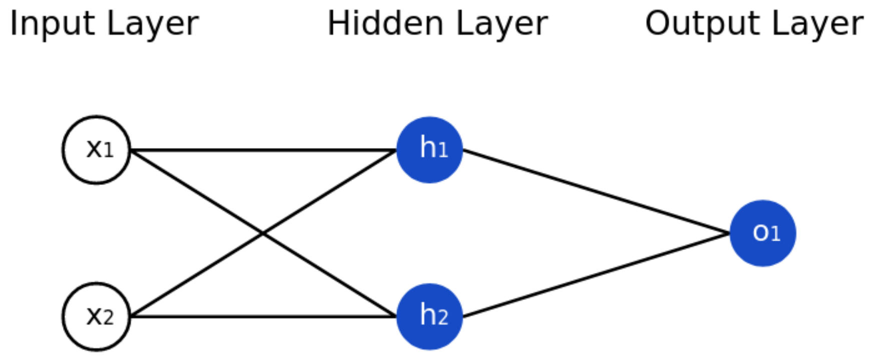
https://victorzhou.com/blog/intro-to-neural-networks/



Harry Li, Ph.D.

Python Code for Feed Forward NN

https://victorzhou.com/blog/intro-to-neural-networks/




```
  def __init__(self):
    weights = np.array([0, 1])
    bias = 0

    # The Neuron class here is from the previous section
    self.h1 = Neuron(weights, bias)
    self.h2 = Neuron(weights, bias)
    self.o1 = Neuron(weights, bias)

  def feedforward(self, x):
    out_h1 = self.h1.feedforward(x)
    out_h2 = self.h2.feedforward(x)

    # The inputs for o1 are the outputs from h1 and h2
    out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

    return out_o1

network = OurNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x))
```

Data Set and Prepare for Training

https://victorzhou.com/blog/intro-to-neural-networks/

Example: Collecting data for training

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

→

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

Note: to reduce the mean value of the data, so it will have balanced distribution for both positive and negative side, it is good for the activation function to handle

Signs	Mij	Mpq	Gender
V1	133	65	Stop
V2	160	72	Right
V3	152	70	Right
V4	120	60	Stop

Define Loss Function

https://victorzhou.com/blog/intro-to-neural-networks/

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

The Mean Square Error Function as a loss function, very often it is also defined as objective function, e.g., minimize the loss function becomes objective function, as shown below step by step.

$$\downarrow$$

$$y_{true} - y_{pred}$$

$$\downarrow$$

$$(y_{true} - y_{pred})^2$$

$$\downarrow$$

$$\sum_{i=1}^{n}(y_{true} - y_{pred})^2 \longrightarrow \text{Exp}\left[\sum_{i=1}^{n}(y_{true} - y_{pred})^2\right]$$

Define Loss Function

https://victorzhou.com/blog/intro-to-neural-networks/

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_{true} - y_{pred})^2$$

The Mean Square Error Function as a loss function, very often it is also defined as objective function, e.g., minimize the loss function becomes objective function, as shown below step by step.

```
import numpy as np
def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()
```

$$\downarrow$$

$$y_{true} - y_{pred}$$

$$\downarrow$$

$$(y_{true} - y_{pred})^2$$

$$\downarrow$$

$$\sum_{i=1}^{n} (y_{true} - y_{pred})^2 \longrightarrow \text{Exp}\left[\sum_{i=1}^{n} (y_{true} - y_{pred})^2\right]$$

Compute Loss Function

https://victorzhou.com/blog/intro-to-neural-networks/

Example: Given the following y_{true} and y_{pred}, compute the MSE

Name	y_{true}	y_{pred}	(y_{true} - y_{pred})^2
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

$$\mathrm{MSE} = \frac{1}{4}(1 + 0 + 0 + 1) = \boxed{0.5}$$

```
import numpy as np
def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()
y_true = np.array([1, 0, 0, 1])
y_pred = np.array([0, 0, 0, 0])
print(mse_loss(y_true, y_pred)) # 0.5
```

Loss Function and Learning

https://victorzhou.com/blog/intro-to-neural-networks/

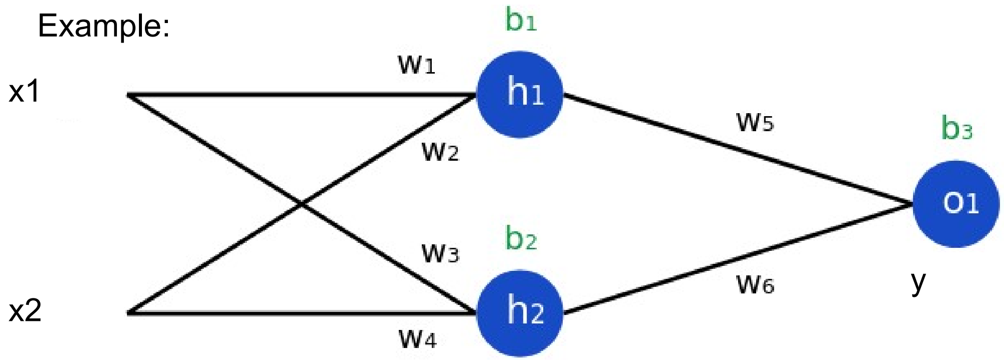
Loss function is a very important function to realize training, and to link the minimization of error to the NN weights.

Define a loss function as a function of NN parameters, e.g., weights and bias

$$L(w\_i , b\_j) = \sum_{i=1}^{n} (y_{true} - y_{pred})^2 \qquad (1)$$

Note y_{pred} is a function of NN parameters w_i, so it can be written as y_{pred} (w_i, b_j).

Example:



From the NN given in this example

We can write loss function with respect to the parameters as follows

L(w_i , b_j) , for I = 1,...,6 and for j = 1,2,3. So from Fig. 1 we have:

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3) \qquad (2)$$

Where h1 can be written as

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1) \qquad (3)$$

Note: we will introduce more generalized notations from equations (1) to (3), for now we will stay with this notation

Remark 1: To minimize loss function with respect to the variables w and b according

Minimize Loss Function

https://victorzhou.com/blog/intro-to-neural-networks/

Minimize loss function by taking partial derivatives with respect to all the parameters w and b, for example

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1} \quad (1)$$

Example: find each partial derivative based on equation (1), then put them together we have

Then update the parameters one by one using the following formula

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1} \quad (2)$$

Note: for the activation function S(x), we have

$$\boxed{S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}} \quad (3)$$

Its derivative is

$$\boxed{S'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x) * (1 - f(x))}$$

Derivative of the Sigmoid function, based on quotient rule (part of chain rule)

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

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{d}{dx}\left(f(x) \cdot \frac{1}{g(x)}\right)$$
$$= f'(x) \cdot \frac{1}{g(x)} + f(x) \cdot \frac{d}{dx}\left(\frac{1}{g(x)}\right)$$

Minimize Loss Function

https://victorzhou.com/blog/intro-to-neural-networks/

Minimize loss function by taking partial derivatives with respect to all the parameters w and b, for example

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1} \quad (1)$$

Example: find each partial derivative based on equation (1), then put them together we have

$$\frac{\partial y_{pred}}{\partial h_1} = w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)$$ = -0.925

$$\frac{\partial y_{pred}}{\partial h_1} = w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)$$ = -0.249

$$\frac{\partial h_1}{\partial w_1} = x_1 * f'(w_1 x_1 + w_2 x_2 + b_1)$$ = -0.0904

$$\frac{\partial L}{\partial w_1} = -0.952 * 0.249 * -0.0904$$ = -0.0214

Note: for the activation function S(x), we have

$$\boxed{S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}} \quad (3)$$

Its derivative is

$$\boxed{S'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x) * (1 - f(x))}$$

Derivative of the Sigmoid function, based on quotient rule (part of chain rule)

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

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{d}{dx}\left(f(x)\cdot\frac{1}{g(x)}\right)$$
$$= f'(x)\cdot\frac{1}{g(x)} + f(x)\cdot\frac{d}{dx}\left(\frac{1}{g(x)}\right)$$

Gradient Steepest Descent Algorithm

https://github.com/hualili/opencv/blob/master/deep-learning-2020S/20-2021S-4gradient-descent-final-2021-2-8.pdf

Lecture Note 1 on Gradient Descent

Harry Li, Ph.D.
Department of Computer Engineering, College of Engineering
San Jose State University, San Jose, CA 95192, USA
Email†: harry.li@ctione.com

***Abstract*—In this lecture note, we give a gradient descent example for its applications in Neural Networks (NN), e.g., the concept of the negative gradient -∇f follows the direction of steepest descent.**

I. Introduction

In this lecture note, we give a gradient descent example for Neural Networks (NN) applications. In particular, the basic concept of the negative gradient -∇f follows the direction of steepest descent of a given function f which can be an error function.

II. Partial Derivative vs. Gradient

Given a scalar-valued multivariable functions, e.g. the function with a multidimensional input $x_1, x_2, ..., x_n$, and a one-dimensional output as $y = f(x_1, x_2, ..., x_)$, where $f : R^n \rightarrow R$. The partial derivative of $f(x_1, x_2, ..., x_n)$ with respect to x_i for $i = 1, 2, ..., n$:

$$\frac{\partial f}{\partial x_i} = \lim_{\delta x \rightarrow 0} \frac{f(x_1, .., x_i + \delta x_i, .., x_n) - f(x_1, ..., x_i, ..., x_n)}{\delta x_i} \tag{1}$$

IV. Gredient Steepest Descent for Minimization

Now, let's define f as an error function, and we would like to minimize it. So we can try to change its inputs (x_1, x_2) by iteration steps:

$$(x_1^{k+1}, x_2^{k+1}) = (x_1^k, x_2^k) + (-\eta \nabla f) \tag{5}$$

which will reduce the function value f. To verify this, write f in terms of Taylar expansion as follows

$$f(x_1, x_2) \simeq f(a, b) + \frac{\partial f}{\partial x_1}(x_1 - a) + \frac{\partial f}{\partial x_2}(x_2 - b) \tag{6}$$

use simplified notation for the partial derivative f_{x_1} and f_{x_2}, we have

$$f(x_1, x_2) \simeq f(a, b) + f_{x_1}(a, b) * (x_1 - a) + f_{x_2}(a, b) * (x_2 - b) \tag{7}$$

we want to update (x_1^k, x_2^k) to (x_1^{k+1}, x_2^{k+1}) such that $f(x_1^{k+1}, x_2^{k+1}) < f(x_1^k, x_2^k)$. From equation (6), replace (x_1, x_2) by (x_1^{k+1}, x_2^{k+1}), and let $(x_1^k, x_2^k) = (a, b)$, so we have