Manual Neural Network

This is really cool, and it's something I've wanted to do. I've got this and several other ways to do a similar thing. This one gets done first. It's going to mimic the TensorFlow API. When I get back to TensorFlow, I should have a better understanding.

From Jose

> In this notebook we will manually build out a neural network that mimics the TensorFlow API. This will greatly help your understanding when working with the real TensorFlow!

Some Info About super() and Object Oriented Programming in General

In [1]:
```
class SimpleClassLecture0():

    def __init__(self):
        print("hello")
    ##endof:  __init__(self)



##endof:  SimpleClassLecture0
```

In [2]:
```
s = "world"
```

In [3]:
```
type(s)
```

Out[3]:
```
str
```

In [4]:
```
# s.<then press [Tab]>
# Gives a list of methods
```

In [5]:
```
x0 = SimpleClassLecture0
```

In [6]:
```
x0 # what we get without the parentheses - __init__ doesn't get called
```

Out[6]:
```
__main__.SimpleClassLecture0
```

In [7]:
```
x0 = SimpleClassLecture0()
```

```
hello
```

In [8]:
```
x0 # Instance of SimpleClassLecture and where it exists in memory
```

Out[8]:
```
<__main__.SimpleClassLecture0 at 0x1668d7c2c50>
```

In [9]:
```
class SimpleClassLecture1():

    def __init__(self):
        print("hello")
    ##endof:  __init__(self)

    def yell(self):
        print("YELLING")
    ##endof:  yell(self)

##endof:  SimpleClassLecture1
```

In [10]:
```
x1 = SimpleClassLecture1()
```

```
hello
```

In [11]:
```
#  I'm going to type 'x1.' then hit [Tab].
#+ it will autocomplete 'x1.yell', after
#+ which I'll add the parenthesis
x1.yell()
```

```
YELLING
```

In [12]:
```
# Now, I'll just type it all out.
x1.yell()
```

```
YELLING
```

In [13]:
```
##  adding in this illustration. These first calls will work fine.
sc = SimpleClassLecture1()
print("--- some separation ---")
sc.yell()
```

```
hello
--- some separation ---
YELLING
```

In [14]:
```
##  continuing with the illustration. This is called
##+ as if it were the lecture notes. It will throw
##+ an error/exception/whatever-you-want-to-call-it
sc_oops = SimpleClassLecture1("Basket Weaving 101")
print("--- some separation ---") # won't execute b/c error before
sc_oops.yell()                   # won't execute b/c error before
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-6c1cabf9d07d> in <module>()
      2 ##+ as if it were the lecture notes. It will throw
      3 ##+ an error/exception/whatever-you-want-to-call-it
----> 4 sc_oops = SimpleClassLecture1("Basket Weaving 101")
      5 print("--- some separation ---") # won't execute b/c error before
      6 sc_oops.yell()                   # won't execute b/c error before

TypeError: __init__() takes 1 positional argument but 2 were given
```

OUTPUT (error) should be

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-87-6c1cabf9d07d> in <module>()
      2 ##+ as if it were the lecture notes. It will throw
      3 ##+ an error/exception/whatever-you-want-to-call-it
----> 4 sc_oops = SimpleClassLecture1("Basket Weaving 101")
      5 print("--- some separation ---") # won't execute b/c error before
      6 sc_oops.yell()                  # won't execute b/c error before

TypeError: __init__() takes 1 positional argument but 2 were given
```

Remember the code:

```
class SimpleClassLecture1():

    def __init__(self):
        print("hello")
    ##endof:  __init__(self)

    def yell(self):
        print("YELLING")
    ##endof:  yell(self)

##endof:  SimpleClassLecture1
```

In [15]:
```
class ExtendedClassLecture0(SimpleClassLecture1):

    def __init__(self):

        print("EXTEND!")

    ##endof:  __init__(self)

##endof:  ExtendedClassLecture0(SimpleClassLecture1)
```

In [16]:
```
y0 = ExtendedClassLecture0()
#  Remember, there's no 'super' call for '__init__'
```

```
EXTEND!
```

In [17]:
```
#  No 'super' with '__init__', but other things work
y0.yell()
```

```
YELLING
```

Now, let's use the `super` keyword.

In [18]:
```
class ExtendedClassLecture1(SimpleClassLecture1):

    def __init__(self):

        super().__init__()
        print("EXTEND!")
    ##endof:  __init__(self)

##endof:  ExtendedClassLecture(SimpleClassLecture)
```

In [19]:
```
y1 = ExtendedClassLecture1()
```

```
hello
EXTEND!
```

In [20]:
```
y1.yell()
```

```
YELLING
```

Here, we're going to add an argument to the `SimpleClass` `__init__` (i.e. its constructor). Since this is the final state in which Jose leaves it, I'm going to use `SimpleClassLecture` instead of continuing with `SimpleClassLecture2`. I'll do similarly with the extended class - using `ExtendedClassLecture` instead of staying with the pattern and using `ExtendedClassLecture2`.

In [21]:
```
class SimpleClassLecture():

    def __init__(self, name):
        print("hello " + name) #  Jose put the space here, which
                               #+ I consider the correct place.
            #+ a minute or so after 1701113954_2023-11-27T123914-0700
    ##endof:  __init__(self)

    def yell(self):
        print("YELLING")
    ##endof:  yell(self)

##endof:  SimpleClassLecture1
```

In [22]:
```
x = SimpleClassLecture("Dave")
```

```
hello Dave
```

In [23]:
```
x.yell()
```

```
YELLING
```

In [24]:
```
class ExtendedClassLecture(SimpleClassLecture):

    def __init__(self):

        super().__init__("Davidushka!")
        print("EXTEND!")

    ##endof:  __init__(self)

##endof:  ExtendedClassLecture(SimpleClassLecture)
```

In [25]:
```
y = ExtendedClassLecture()
```

```
hello Davidushka!
EXTEND!
```

In [26]:
```
y.yell()
```

```
YELLING
```

From the class material

In [27]:
```
class SimpleClass():

    def __init__(self, str_input):
        #  DWB: I'm not fixing his lack of space after "SIMPLE".
        #+      1701111285_2023-11-27T115445-0700
        print("SIMPLE" + str_input)
    ##endof:  __init__(self, str_input)

##endof:  SimpleClass
```

I'll do the same two illustrations.

In [28]:
```
sc = SimpleClass() # will throw an error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-1a19d7d610fd> in <module>()
----> 1 sc = SimpleClass() # will throw an error

TypeError: __init__() missing 1 required positional argument: 'str_input'
```

OUTPUT:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-29-1a19d7d610fd> in <module>()
----> 1 sc = SimpleClass() # will throw an error

TypeError: __init__() missing 1 required positional argument: 'str_input'
```

In [29]:
```
##  This one should work fine, though the lack of a space between
##+ "SIMPLE" and "Basket Weaving 101" - i.e.
##+ "SIMPLEBasket Weaving 101", grates on my nerves a bit. Q&R
sc = SimpleClass("Basket Weaving 101")
```

```
SIMPLEBasket Weaving 101
```

Remember the code (defined in the lecture notes)

```
class SimpleClass():

    def __init__(self, str_input):
        #  DWB: I'm not fixing his lack of space after "SIMPLE".
        #+      1701111285_2023-11-27T115445-0700
        print("SIMPLE" + str_input)
    ##endof:  __init__(self, str_input)

##endof:  SimpleClass
```

In [30]:
```
class ExtendedClassNoSuper(SimpleClass):

    def __init__(self):
        print('EXTENDED')
    ## endof:  __init__(self)

##endof:  ExtendedClassNoSuper
```

In [31]:
```
s = ExtendedClassNoSuper()
```

```
EXTENDED
```

With the output, remember that we *overwrote* the `__init__(self)` method.

What I'll call `ExtendedClass` is building upon the `ExtendedClassNoSuper` code. I could have added `Super` at the end (`ExtendedClassSuper`), or I could have done as the lecture notes did and call both `ExtendedClass` , with one replacing the other. Anyway, `ExtendedClass` will use `super` .

In [32]:
```
#  remember to use 'class' instead of 'def'
#+ (Oops, DWB 1701111919_2023-11-27T120519-0700)

class ExtendedClass(SimpleClass):

    def __init__(self):

        super().__init__(" My String") # Jose puts the space in the string here.
        print('EXTENDED')

    ##endof:  def __init__(self)

##endof:  ExtendedClass
```

In [33]:
```
s = ExtendedClass()
```

```
SIMPLE My String
EXTENDED
```

We've finished learning some OOP stuff - now for the Manual NN

Operation

In [ ]:

Example Operations

Addition

In [ ]:

Multiplication

In [ ]:

Matrix Multiplication

In [ ]:

Placeholders

In [ ]:

Variables

In [ ]:

Graph

In [ ]:

A Basic Graph

Come back and fill this in.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Session

In [ ]:

Traversing Operation Nodes

In [ ]:

In [ ]:

Let's try it!

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Now, some matrix multiplication

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Activation Function

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Sigmoid as an Operation

In [ ]:

Classification Example

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Defining the Perceptron

blah!

Convert to a Matrix Representation of Features

blah! Strong Bad. blah!

Example Point

and blah! again.

In [ ]:

something else

In [ ]:

Using an Example Session Graph

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

That's all for now, folks!

In [ ]: