This notebook is an exercise in the Intro to Deep Learning course. You can reference the tutorial at this link.

Introduction

In the tutorial, we saw how to build deep neural networks by stacking layers inside a `Sequential` model. By adding an *activation function* after the hidden layers, we gave the network the ability to learn more complex (non-linear) relationships in the data.

In these exercises, you'll build a neural network with several hidden layers and then explore some activation functions beyond ReLU. Run this next cell to set everything up!

In [1]:

```
import tensorflow as tf

# Setup plotting
import matplotlib.pyplot as plt

plt.style.use('seaborn-whitegrid')
# Set Matplotlib defaults
plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsize='large',
       titleweight='bold', titlesize=18, titlepad=10)

# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.deep_learning_intro.ex2 import *
```

In the *Concrete* dataset, your task is to predict the compressive strength of concrete manufactured according to various recipes.

Run the next code cell without changes to load the dataset.

In [2]:

```
import pandas as pd

concrete = pd.read_csv('../input/dl-course-data/concrete.csv')
concrete.head()
```

Out[2]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	Co
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	

1) Input Shape

The target for this task is the column `'CompressiveStrength'` . The remaining columns are the features we'll use as inputs.

What would be the input shape for this dataset?

In [3]:

```
# YOUR CODE HERE
input_shape = [len(concrete.columns) -1]

# Check your answer
q_1.check()
```

Correct

In [4]:

```
# Lines below will give you a hint or solution code
#q_1.hint()
# q_1.solution()
```

2) Define a Model with Hidden Layers

Now create a model with three hidden layers, each having 512 units and the ReLU activation. Be sure to include an output layer of one unit and no activation, and also `input_shape` as an argument to the first layer.

In [5]:

```
from tensorflow import keras
from tensorflow.keras import layers

# YOUR CODE HERE
model = keras.Sequential([
    # the hidden ReLU layers
    layers.Dense(units=512, activation='relu', input_shape=input_shape),
    layers.Dense(units=512, activation='relu'),
    layers.Dense(units=512, activation='relu'),
    # the linear output layer
    layers.Dense(units=1),
])

# Check your answer
q_2.check()
```

```
2022-12-18 05:21:47.519878: I tensorflow/core/common_runtime/process_util.cc:146] Creati
ng new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_thr
eads for best performance.
```

Correct

In [6]:

```
# Lines below will give you a hint or solution code
#q_2.hint()
#q_2.solution()
```

3) Activation Layers

Let's explore activations functions some.

The usual way of attaching an activation function to a `Dense` layer is to include it as part of the definition with the `activation` argument. Sometimes though you'll want to put some other layer between the `Dense` layer and its activation function. (We'll see an example of this in Lesson 5 with *batch normalization*.) In this case, we can define the activation in its own `Activation` layer, like so:

```
layers.Dense(units=8),
layers.Activation('relu')
```

This is completely equivalent to the ordinary way: `layers.Dense(units=8, activation='relu')` .

Rewrite the following model so that each activation is in its own `Activation` layer.

In [7]:

```
### YOUR CODE HERE: rewrite this to use activation layers
model = keras.Sequential([
    layers.Dense(32, input_shape=[8]),
    layers.Activation('relu'),
    layers.Dense(32),
    layers.Activation('relu'),
    layers.Dense(1),
])

# Check your answer
q_3.check()
```

Correct

In [8]:

```
# Lines below will give you a hint or solution code
#q_3.hint()
# q_3.solution()
```

Optional: Alternatives to ReLU

There is a whole family of variants of the `'relu'` activation -- `'elu'` , `'selu'` , and `'swish'` , among others -- all of which you can use in Keras. Sometimes one activation will perform better than another on a given task, so you could consider experimenting with activations as you develop a model. The ReLU activation tends to do well on most problems, so it's a good one to start with.

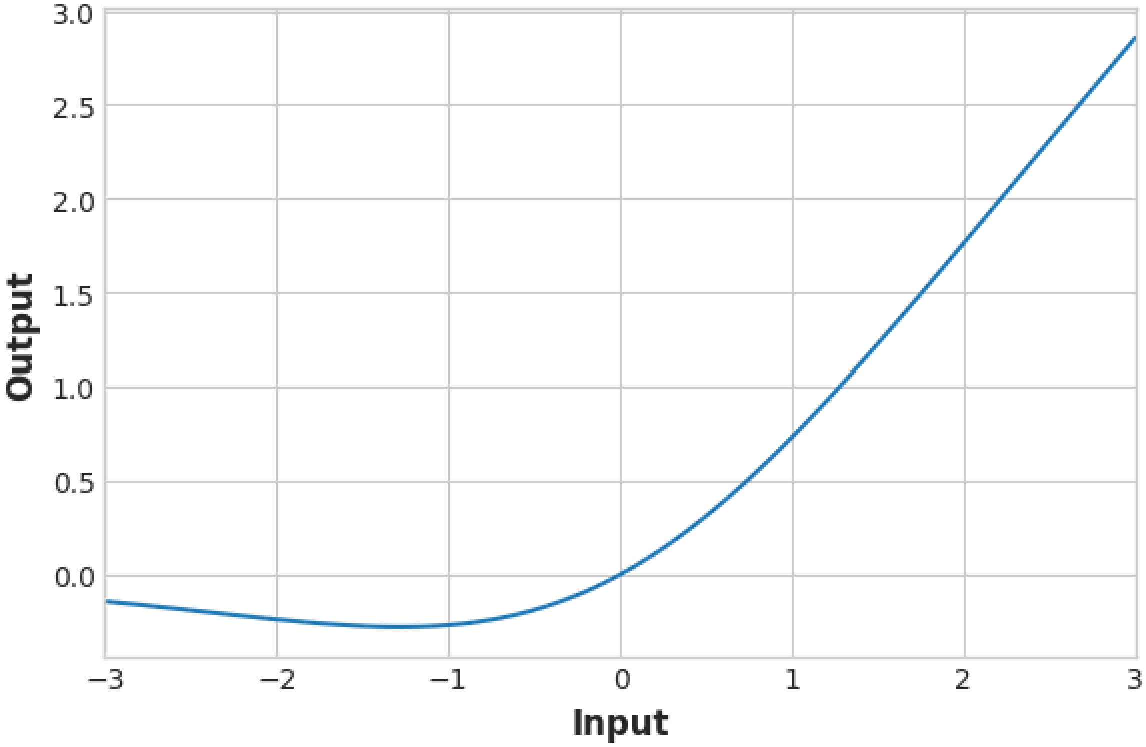
Let's look at the graphs of some of these. Change the activation from `'relu'` to one of the others named above. Then run the cell to see the graph. (Check out the documentation for more ideas.)

In [9]:

```
# YOUR CODE HERE: Change 'relu' to 'elu', 'selu', 'swish'... or something else
activation_layer = layers.Activation('swish')

x = tf.linspace(-3.0, 3.0, 100)
y = activation_layer(x) # once created, a layer is callable just like a function

plt.figure(dpi=100)
plt.plot(x, y)
plt.xlim(-3, 3)
plt.xlabel("Input")
plt.ylabel("Output")
plt.show()
```



Keep Going

Now move on to Lesson 3 and **learn how to train neural networks** with stochastic gradient descent.

Have questions or comments? Visit the course discussion forum to chat with other learners.