EMERGING METHODS FOR EARLY DETECTION OF FOREST FIRES

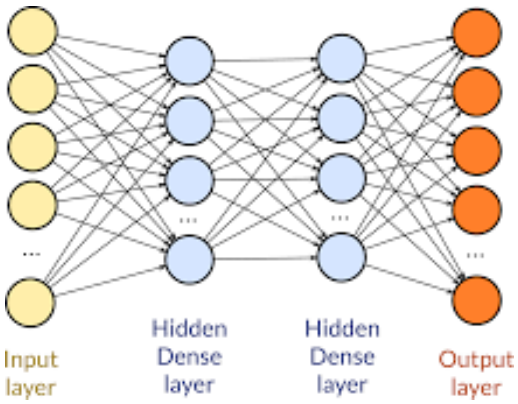
MODEL BUILDING

INITIALIZING THE MODEL

Date	28 October 2022
Team ID	PNT2022TMID03761
Project Name	Emerging Methods for Early Detection of Forest Fires

INITIALIZING THE MODEL:

Keras has 2 ways to define a neural network:



Sequential:

- The core idea of ***Sequential API*** is simply arranging the Keras layers in a sequential order and so, it is called *Sequential API*. Most of the ANN also has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.

- A ANN model can be created by simply calling **Sequential()** API as specified below −

```
from keras.models import Sequential
model = Sequential()
```

- Add layers:

To add a layer, simply create a layer using Keras layer API and then pass the layer through add() function as specified below −

```
from keras.models import Sequential

model = Sequential()
input_layer = Dense(32, input_shape=(8,)) model.add(input_layer)
hidden_layer = Dense(64, activation='relu'); model.add(hidden_layer)
output_layer = Dense(8)
model.add(output_layer)
```

Here, we have created one input layer, one hidden layer and one output layer.

Access the model

Keras provides few methods to get the model information like layers, input data and output data. They are as follows −

- ***model.layers*** − Returns all the layers of the model as list.

```
>>> layers = model.layers
>>> layers
[
  <keras.layers.core.Dense object at 0x000002C8C888B8D0>,
  <keras.layers.core.Dense object at 0x000002C8C888B7B8>
```

```
  <keras.layers.core.Dense object at 0x 000002C8C888B898>
]
```

- ***model.inputs*** − Returns all the input tensors of the model as list.

```
>>> inputs = model.inputs
>>> inputs
[<tf.Tensor 'dense_13_input:0' shape=(?, 8) dtype=float32>]
```

- ***model.outputs*** − Returns all the output tensors of the model as list.

```
>>> outputs = model.outputs
>>> outputs
<tf.Tensor 'dense_15/BiasAdd:0' shape=(?, 8) dtype=float32>]
```

- ***model.get_weights*** − Returns all the weights as NumPy arrays.
- ***model.set_weights(weight_numpy_array)*** − Set the weights of the model

- Sequential- A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor**.**
- Keras is an API that gets well with neural network models related to artificial intelligence and machine learning so is the keras sequential which deals with ordering or sequencing of layers within a model.

- The core idea of Sequential API is simply arranging the Keras layers in a sequential order and so, it is called Sequential API. Most of the ANN also

has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.

- In Keras, "dense" usually refers to a single layer, whereas "sequential" usually refers to an entire model, not just one layer. So I'm not sure the comparison between "Dense vs. Sequential" makes sense. Sequential refers to the way you build models in Keras using the sequential api.

- A CNN can be instantiated as a Sequential model because each layer has exactly one input and output and is stacked together to form the entire network.

A Sequential model is not appropriate when:

- Your model has multiple inputs or multiple outputs
- Any of your layers has multiple inputs or multiple outputs
- You need to do layer sharing
- You want non-linear topology (e.g. a residual connection, a multi-branch model.

Function API:

The Keras functional API provides a more flexible way for defining models.

- It specifically allows you to define multiple input or output models as well as models that share layers. More than that, it allows you to define ad hoc acyclic network graphs.

- Models are defined by creating instances of layers and connecting them directly to each other in pairs, then defining a Model that specifies the layers to act as the input and output to the model.

Function API- the Keras functional API is the way to go for defining complex models, such as multi-output models, directed acyclic graphs, or models with shared layers. This guide assumes that you are already familiar with the Sequential model.

The Keras functional API is a way to create models that are more flexible than the tf.keras .Sequential API. The functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs.

The main idea is that a deep learning model is usually a directed acyclic graph (DAG) of layers. So the functional API is a way to build graphs of layers. The functional API in Keras is an alternate way of creating models that offers a lot more flexibility, including creating more complex models.

Here, Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method.

Now, will initialize our model. Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use

the Sequential constructor to create a model, which will then have layers added to it using the add () method.

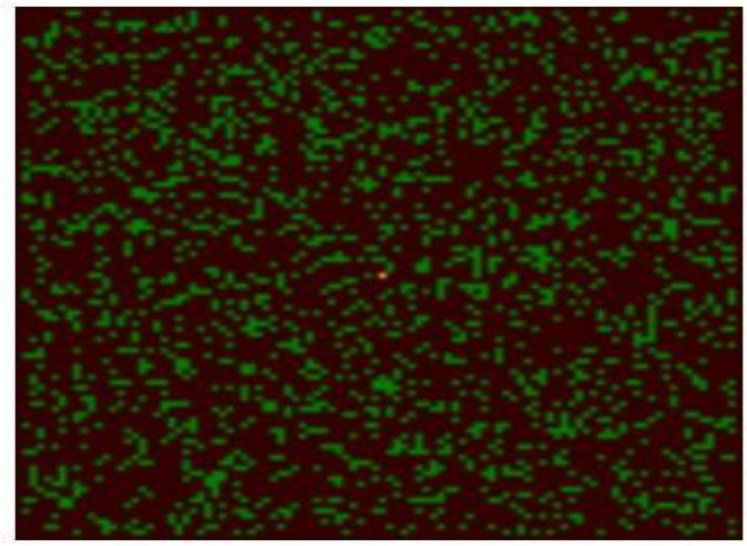
Now, will initialize our model.

```
#intializing the model
model =Sequential()
```

MODEL OVERVIEW:

A simple model of a forest fire is defined as a two-dimensional cellular automaton on a grid of cells which take one of three states: empty, occupied by a tree, or burning. The automaton evolves according to the following rules which are executed simultaneously for every cell at a given generation.

1. A burning cell turns into an empty cell
2. A cell occupied by a tree becomes a burning cell if any of its eight neighbouring cells are burning
3. A cell occupied by a tree becomes burning with a probabilty ff (even if none of its neighbouring cells are burning), as though struck by lightning
4. An empty cell becomes occupied by a tree with probability pp.



The model is interesting as a simple dynamical system displaying self-organized criticality. The following Python code simulates a forest fire using this model for probabilities p=0.05p=0.05, f=0.001f=0.001. A Matplotlib animation is used for visualization.

The simulation isn't very realistic for small ff: fires in a dense forest tend to expand out in a square pattern because diagonally-adjacent trees catch fire as readily as orthogonally-adjacent trees. This can be improved by assigning a probability of less than 1 for these trees to ignite. It isn't obvious what probability to assign, but I use 0.573 below. This is the ratio B/AB/A, where BB is the area of overlap of a circle of radius 3232 at (0,0) (the tree on fire) with one of radius 1212 at (0,2–√)(0,2) (blue, the diagonally-adjacent tree), and AA is the overlap of this first circle with one of radius 1212 at (0,1)(0,1) (red, the orthogonally-adjacent tree). AA is equal to the "area" of this adjacent tree, since its circle is contained in the first.