Keras

Introdction

Alok

- Consider the following eight steps to create deep learning model in Keras −
- Loading the data
- Preprocess the loaded data
- Definition of model
- Compiling the model
- Fit the specified model
- Evaluate it
- Make the required predictions
- Save the model

Preprocess and load data-data is the key for the working of neural network and we need to process it before feeding to the neural network.

In this step, we will also visualize data which will help us to gain insight into the data.

Define model- Now we need a neural network model.

This means we need to specify the number of hidden layers in the neural network and their size, the input and output size.

Loss and optimizer- Now we need to define the loss function according to our task.

We also need to specify the optimizer to use with learning rate and other hyperparameters of the optimizer.

Fit model- This is the training step of the neural network. Here we need to define the number of epochs for which we need to train the neural network.

Step 1: Choose a Model

Step 2: Training Phase

Step 4: Evaluation Phase

Create a Model

```
model = keras.models.Sequential()
model.add(keras.layers.Dense())
…. add more layers …
model.compile(loss='mean_squared_error',
optimizer='adam')
```

Keras Sequential Model API

- The easiest way to build neural networks in Keras
- Create an empty sequential model object and then add layers to it in sequence

Building the Model

- Every Keras model is either built using the Sequential class, which represents a linear stack of layers, or the functional Model class, which is more customizeable.

Models in Keras are defined as a sequence of layers.

- We create a Sequential model and add layers one at a time until we are happy with our network architecture.
- We'll be using the simpler Sequential model, since our network is indeed a linear stack of layers.

Keras Sequential Model API

```
model = keras.models.Sequential()
model.add(Dense(32, input_dim=9))
model.add(Dense(128))
model.add(Dense(1))
```

Customizing Layers

```
model.add(Dense(number_of_neurons, activation='relu'))
```

```
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Initializing the ANN

For building the Neural Network layer by layer

***from* keras.models *import* Sequential**

model = Sequential()

Sequential specifies to keras that we are creating model sequentially and the output of each layer we add is input to the next layer we specify.

use the add function to add layers.

model.add is used to add a layer to our neural network. We need to specify as an argument what type of layer we want.

The **Dense** is used to specify the fully connected layer.

***from* keras.layers *import* Dense**

The first thing to get right is to ensure the input layer has the right number of input features.

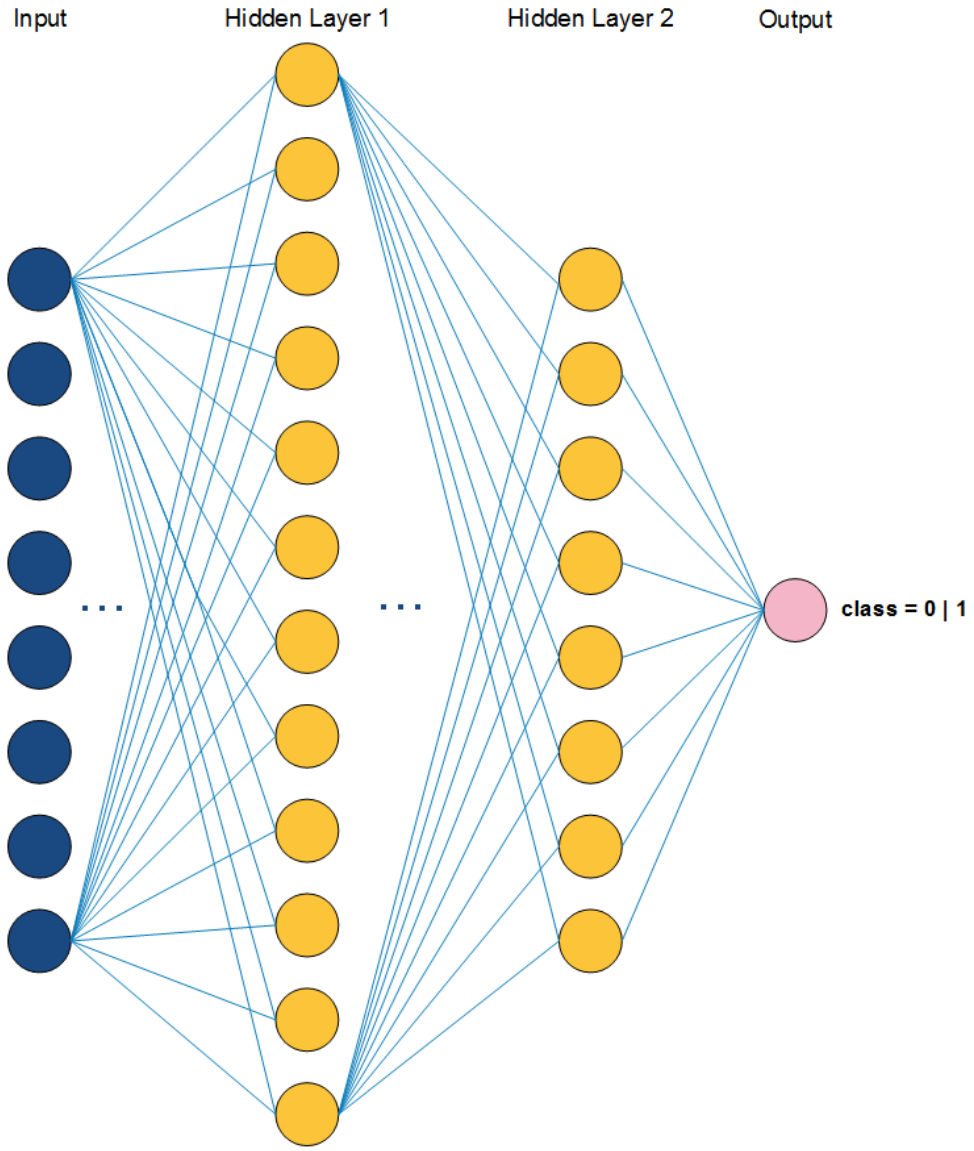
This can be specified when creating the first layer with the **input_dim** argument.

We can specify the activation function using the **activation** argument.

- input_dim is the number of dimensions of the features, in your case that is just 3. The equivalent notation for input_shape, which is an actual dimensional shape, is (3,)
- https://stackoverflow.com/questions/44747343/keras-input-explanation-input-shape-units-batch-size-dim-etc

model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))

It means 8 input parameters, with 12 neurons in the FIRST hidden layer.



```
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Layer Options

Customizable layer settings

- Layer activation function
- Initializer function for node weights
- Regularization function for node weights

But the default settings are a good start!

Other Types of Layers Supported

Convolutional layers

- Example: `keras.layers.convolutional.Conv2D()`

Recurrent layers

- Example: `keras.layers.recurrent.LSTM()`

Keras Sequential Model API

```
model = keras.models.Sequential()
model.add(Dense(32, input_dim=9))
model.add(Dense(128))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Import `Sequential` from `keras.models`
- from keras.models import Sequential

Import `Dense` from `keras.layers`
- from keras.layers import Dense

Initialize the model
- model = Sequential()

Add input layer
- model.add(Dense(64, input_dim=12, activation='relu'))

Add output layer
- model.add(Dense(1))

Compile a model

- Before we can start training our model we need to configure the learning process.
- For this, we need to specify an optimizer, a loss function and optionally some metrics like accuracy.
- The **loss function** is a measure on how good our model is at achieving the given objective.
- An **optimizer** is used to minimize the loss(objective) function by updating the weights using the gradients.

Now we need to specify the loss function and the optimizer. It is done using compile function in keras.

- model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

- Here loss is cross entropy loss as discussed earlier. Categorical_crossentropy specifies that we have multiple classes.
- The optimizer is Adam.
- Metrics is used to specify the way we want to judge the performance of our neural network.

Steps

	from keras.models import Sequential
	from keras.layers import Dense, Activation
	dims = X_train.shape[1]
	print(dims, 'dims')
	print("Building model...")
	nb_classes = Y_train.shape[1]
	print(nb_classes, 'classes')
	model = Sequential()
	model.add(Dense(nb_classes, input_shape=(dims,), activation='sigmoid'))
	model.add(Activation('softmax'))
	model.compile(optimizer='sgd', loss='categorical_crossentropy')
	model.fit(X_train, Y_train)

Compiling the Model

- Before we can begin training, we need to configure the training process. We decide 3 key factors during the compilation step:

The **optimizer:**

- We'll stick with a pretty good default: the Adam gradient-based optimizer.
- Keras has many other optimizers you can look into as well.

The **loss function:**

- Since we're using a Softmax output layer, we'll use the Cross-Entropy loss. Keras distinguishes between binary_crossentropy (2 classes) and categorical_crossentropy (>2 classes), so we'll use the latter. See all Keras losses.

A list of **metrics**.

- Since this is a classification problem, we'll just have Keras report on the **accuracy** metric.

Training Phase

```
model.fit(training_data, expected_output)
```

Testing Phase

```
error_rate = model.evaluate(testing_data,
expected_output)
```

Testing Phase

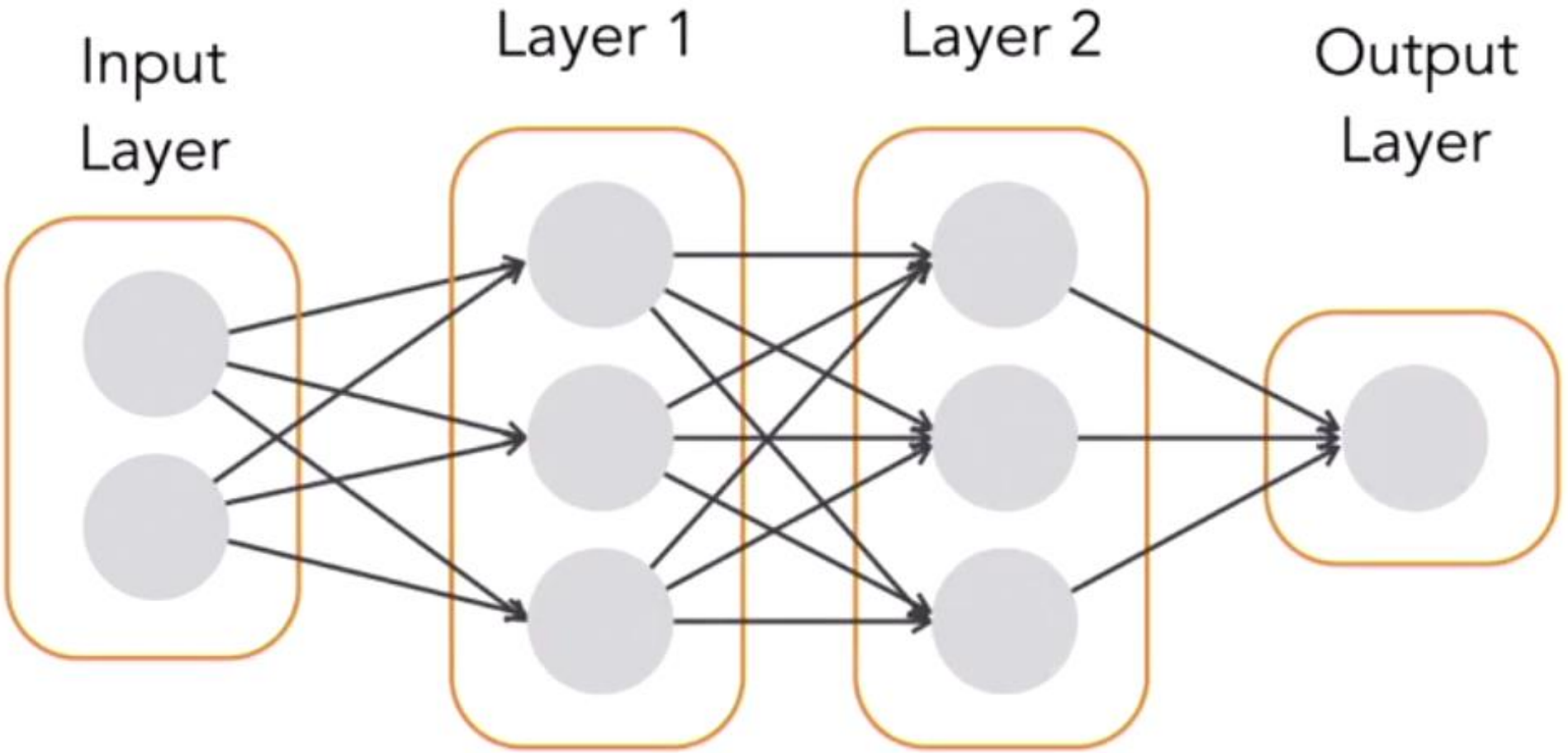
```
error_rate = model.evaluate(testing_data,
expected_output)

model.save(“trained_model.h5”)
```

Evaluation Phase

```
model = keras.models.load_model('trained_model.h5')

predictions = model.predict(new_data)
```


Input Layer
Layer 1
Layer 2
Output Layer

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

data_dim = 16
timesteps = 8
num_classes = 10

# expected input data shape: (batch_size, timesteps, data_dim)
model = Sequential()
model.add(LSTM(32, return_sequences=True,
               input_shape=(timesteps, data_dim)))  # returns a sequence of vectors of dimension 32
model.add(LSTM(32, return_sequences=True))  # returns a sequence of vectors of dimension 32
model.add(LSTM(32))  # return a single vector of dimension 32
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Generate dummy training data
x_train = np.random.random((1000, timesteps, data_dim))
y_train = np.random.random((1000, num_classes))

# Generate dummy validation data
x_val = np.random.random((100, timesteps, data_dim))
y_val = np.random.random((100, num_classes))

model.fit(x_train, y_train,
          batch_size=64, epochs=5,
          validation_data=(x_val, y_val))
```

- https://intellipaat.com/community/1635/keras-input-explanation-inputshape-units-batchsize-dim-etc