DL

DL

Keras

DL

AI ML DL

AI ML DL

AI

if-else

ML

ML

ML DL

ML

DL ML

DL DNNs

ML

ML DL

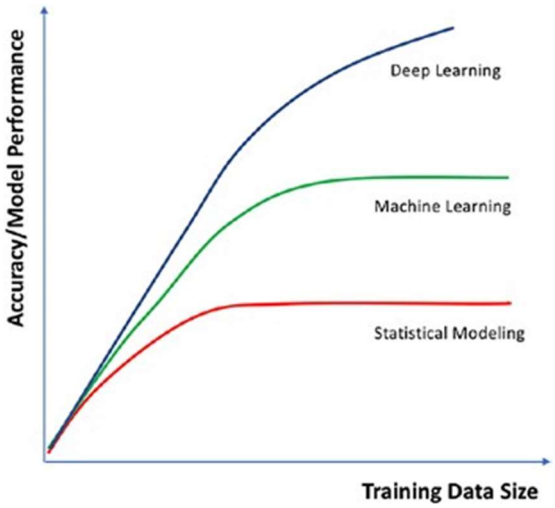
DL

ML DL

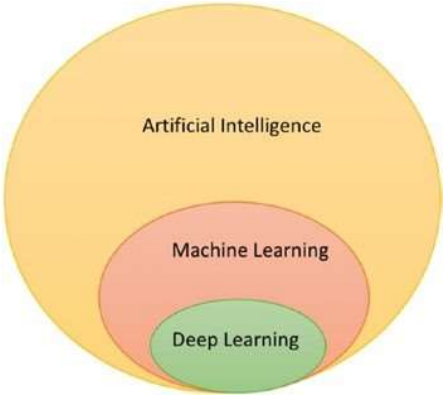
ML

DL

ML



ML



AL

ML

ML DL

DL

DL

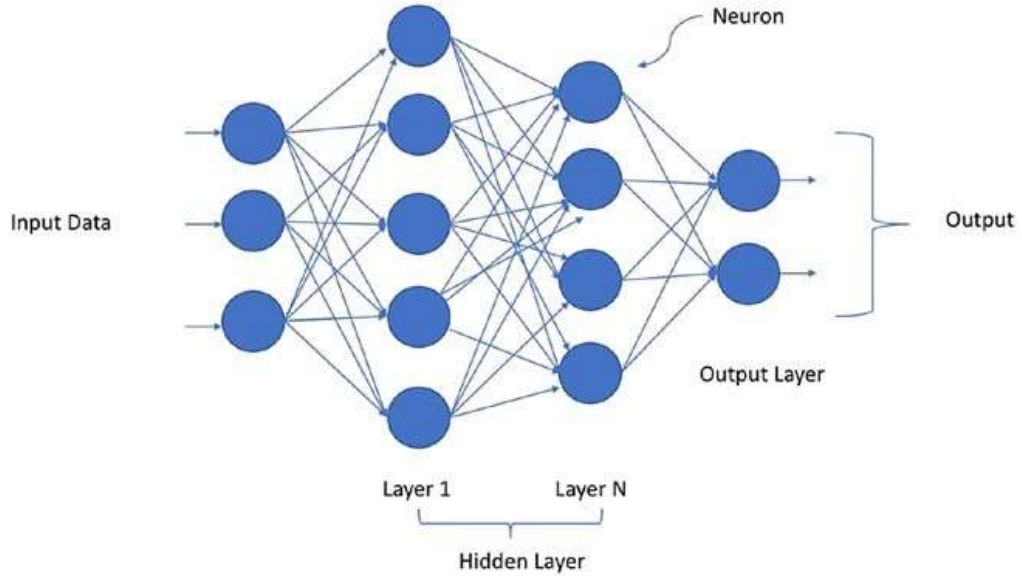
iOS

DL

DL

DL

DL



" "

[1] **backpropagation**

DL

DL

DL

DL

DL

DL

[2] activation functions
[3] Optimization algorithms

1000 C DNN

15 10

DL

DL

DL

DL

DL Theano

Theano MILA

MILA 2017 2007

http://deeplearning.net/software/theano/
https://github.com/Theano/Theano

Torch

Lua DL ML Torch

Clement Farabet Koray Kavukcuoglu Ronan Collobert

http://torch.ch

PyTorch

AI DL ML PyTorch

DL Python Torch PyTorch

DL PyTorch

https://pytorch.org

MxNet

MIT NUS NYU CMU "maximize" "mix" "mix-net" MxNet

Azure AWS

https://mxnet.apache.org

TensorFlow

DL DL TensorFlow

GPU CPU Google

2015

www.tensorflow.org

DL

Microsoft CNTK Caffe

PaddlePaddle Chainer

https://blogs.technet.microsoft.com/ machinelearning/2018/03/14/comparing-deeplearning-frameworks-a-rosetta-stone-approach

DL

DL

DL

C Python

DL

DL

Keras DL DL

Keras Lasagne Gluon

Keras	Theano	Lasagne	MxNet	Gluon
			CNTK	MxNet Theano TensorFlow

DL API Keras

15

Keras

API DL

Keras

API

DL API Keras TensorFlow Keras

TensorFlow Keras API TensorFlow

Keras

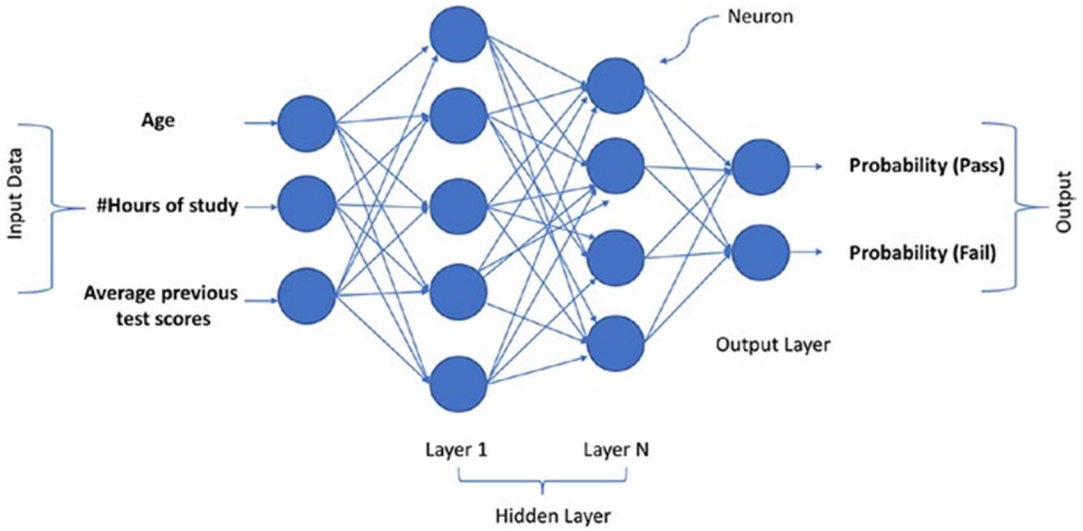
https://keras.io

Keras

DL

DL Keras

DNN



DNN " DL "

DNN

100

Keras

```
#Import required packages
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
# Getting the data ready
# Generate train dummy data for 1000 Students and dummy test for 500
#Columns :Age, Hours of Study &Avg Previous test scores
np.random.seed(2018). #Setting seed for reproducibility
train_data, test_data = np.random.random((1000, 3)), np.random.random((500, 3))

#Generate dummy results for 1000 students : Whether Passed (1)or Failed (0)
labels = np.random.randint(2, size=(1000, 1))
#Defining the model structure with the required layers,
# of neurons, activation function and optimizers
model = Sequential()
model.add(Dense(5, input_dim=3, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
#Train the model and make predictions
```

```
model.fit(train_data, labels, epochs=10, batch_size=32)
#Make predictions from the trained model
predictions = model.predict(test_data)
```

numpy

500 1000

DNN

Keras **DL**

DNN

Keras

DNN 1 "DL "

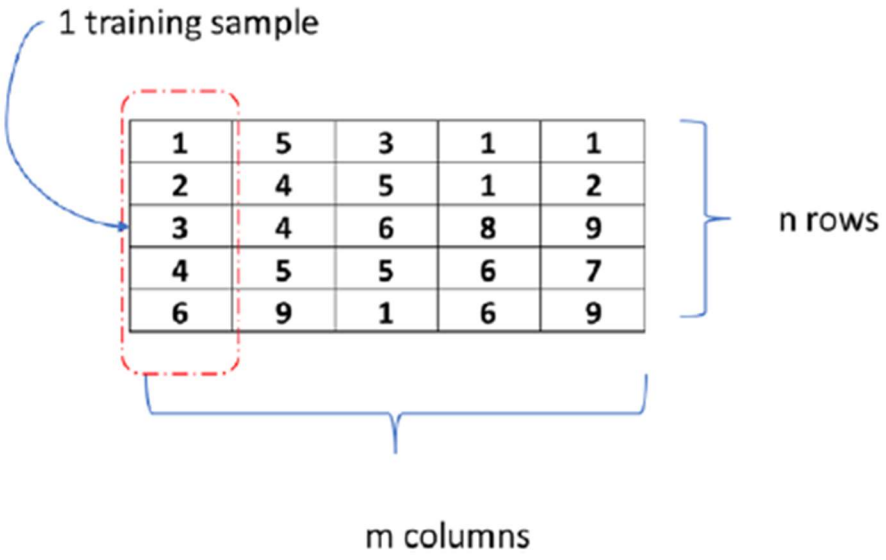
DL

"

n

m

2 dimensional tensor with shape (m x n)



n n

DL

m m

DL

" " " " " "

0 1 0

" " 1 " "

n

RGB
n
512 512 100
100 3 512 512 4

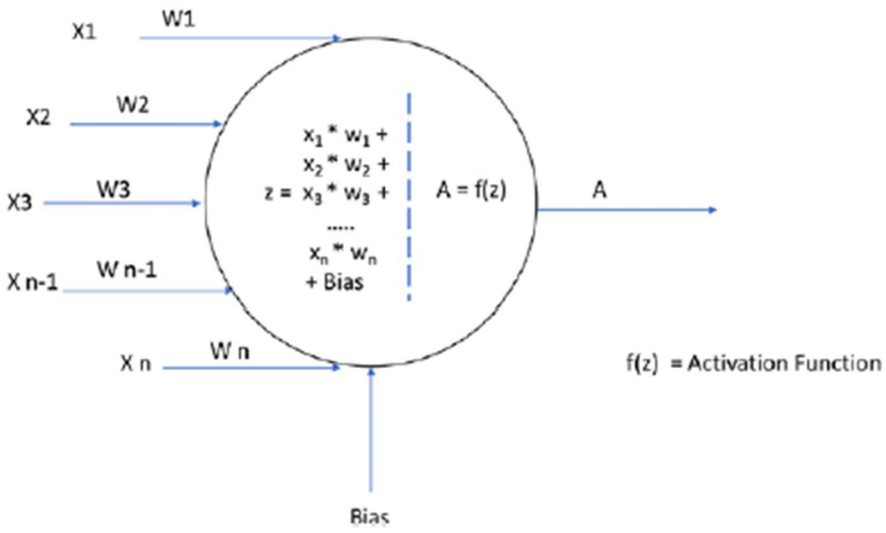
1 0
1 0

DNN

f(z)

<

A Single Neuron



z

z z

Infinity

Infinity

0

DNN

DNN

n

ReLU

1 0

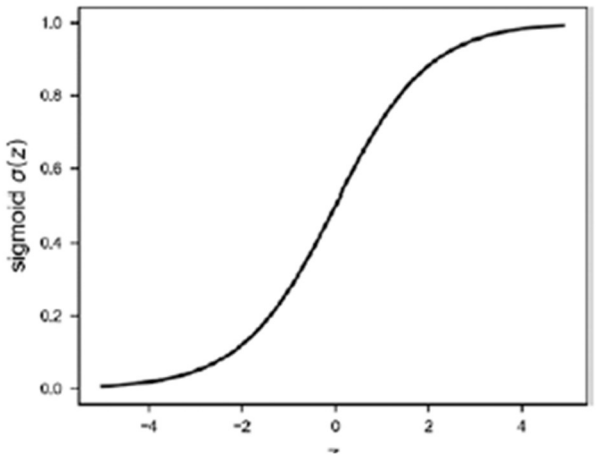
0

1

keras Keras

import activations.sigmoid(x)

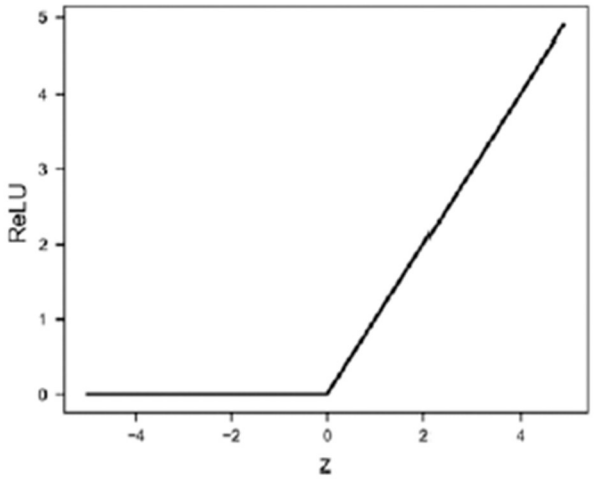
keras.activations.sigmoid



ReLU

f(z) = max(0,z) ReLU

0



ReLU

z 0

0

0

Leaky ReLU

Leaky ReLU

f(z) = z ; when z >0
f(z) = ∝z ; when z<0 and where ∝ is a parameter that is defined as a small constant, say 0.005

keras.layers.LeakyReLU(X, alpha=0.0, max_value=None).

DNN

elu selu elu swish tanh

DNN

15 10

ReLU

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential()
model.add(Dense(10, input_dim=15))
model.add(Activation('relu'))
```

DNN

DL

DL

Keras

- Dense Layer

DNN

15 2 1 3 2 5 1

Keras " " 3 5

```
keras.layers.Dense(units, activation=None, use_bias=True,
kernel_initializer='glorot_uniform',
bias_initializer='zeros',
kernel_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
kernel_constraint=None,
bias_constraint=None)
```

Keras

input_dim

10 input_dim 1000 10

5 1

input_dim = 10 10

```
model = Sequential()
model.add(Dense(5,input_dim=10,activation = "sigmoid"))
model.add(Dense(1,activation = "sigmoid"))
```

DL dropout

0

5

Keras

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

DL

```
model = Sequential()
model.add(Dense(5,input_dim=10,activation = "sigmoid"))
model.add(Dropout(rate = 0.1,seed=100))
model.add(Dense(1,activation = "sigmoid"))
```

Keras

RNN DNN

Keras

Keras

- https://keras.io/layers/embeddings
- https://keras.io/layers/convolutional
- https://keras.io/layers/pooling
- https://keras.io/layers/merge
- https://keras.io/layers/recurrent
- https://keras.io/layers/normalization

Keras

https://keras.io/layers

100 96 90 78 60 56

0 100 1

0.87

0.40 0.13 0.87 1.00

0.05

DL ML

- 9 9 3

81

keras.losses.mean_squared_error(y_actual,y_pred)

•

keras.losses.mean_absolute_error (y_actual, y_pred)

• MAPE – Mean absolute percentage error
keras.losses.mean_absolute_percentage_error
• MSLE – Mean square logarithmic error
keras.losses.mean_squared_logarithmic_error

0 1

C B A

Keras

•

keras.losses.binary_crossentropy(y_actual, y_predicted)

2

n … 2 1

keras.losses.categorical_crossentropy (y_actual, y_predicted)

DNN

Pro Deep Learning with TensorFlow

DL Apress, 2017

DL

Keras

SGD

SGD

Weights = Weights – learning rate * Loss
Where learning rate is a parameter we define in the network architecture.

Say, for learning rate =0.01

keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0,nesterov=False)

_batch

SGD size=1

2

128 64 32

DL Adaptive Moment Estimation Adam

Adam

Weights = Weights – (Momentum and Variance combined)

keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,epsilon=None, decay=0.0, amsgrad=False)

beta_2 beta_1

DL

Keras

- Adagrad
- Adadelta
- RMSProp
- Adamax
- Nadam

DL

Keras

Keras

- keras.metrics.binary_accuracy
- keras.metrics.caetogrical_accuracy
- keras.metrics.sparse_categorical_accuracy

Keras DNN

Keras

" "

ReLU 16 32 DNN

Adam

" "

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential()
model.add(Dense(32, input_dim=10,activation = "relu"))
model.add(Dense(16,activation = "relu"))
model.add(Dense(1,activation = "sigmoid"))
model.compile(optimizer='Adam',loss='binary_crossentropy',
metrics=['accuracy'])
```

20 20 60 60:20:20

%95 n>1MN

%3 %2

Keras

```
model.fit(x_train, y_train, batch_size=64, epochs=3,validation_data=(x_val, y_val))
```

y_train x_train

500 64

64 64

64 500

x_val

y_val

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
# Generate dummy training dataset
np.random.seed(2018)
x_train = np.random.random((6000,10))
y_train = np.random.randint(2, size=(6000, 1))
# Generate dummy validation dataset
x_val = np.random.random((2000,10))
y_val = np.random.randint(2, size=(2000, 1))
# Generate dummy test dataset
x_test = np.random.random((2000,10))
y_test = np.random.randint(2, size=(2000, 1))
#Define the model architecture
```

```
model = Sequential()
model.add(Dense(64, input_dim=10,activation = "relu")) #Layer 1
model.add(Dense(32,activation = "relu")) #Layer 2
model.add(Dense(16,activation = "relu")) #Layer 3
model.add(Dense(8,activation = "relu")) #Layer 4
model.add(Dense(4,activation = "relu")) #Layer 5
model.add(Dense(1,activation = "sigmoid")) #Output
Layer
#Configure the model
model.compile(optimizer='Adam',loss='binary_crossentropy',metri
cs=['accuracy'])
#Train the model
model.fit(x_train, y_train, batch_size=64, epochs=3,
validation_data=(x_val,y_val))
```

```
Train on 6000 samples, validate on 2000 samples
Epoch 1/3
6000/6000 [==============================] - 2s 363us/step - loss: 0.6934 - acc: 0.4972 - val_loss: 0.6932 - val_acc:
0.4945
Epoch 2/3
6000/6000 [==============================] - 0s 58us/step - loss: 0.6933 - acc: 0.4993 - val_loss: 0.6934 - val_acc:
0.4945
Epoch 3/3
6000/6000 [==============================] - 0s 55us/step - loss: 0.6931 - acc: 0.5045 - val_loss: 0.6933 - val_acc:
0.5110
```

DL

Keras

Keras

```
evaluate(x=None, y=None, batch_size=None, verbose=1, sample_
weight=None, steps=None)
```

y x

Keras

```
print(model.evaluate(x_test,y_test))
[0.6925005965232849, 0.521]
```

metrics_names

```
print(model.metrics_names)
['loss', 'acc']
```

%52

```
#Make predictions on the test dataset and print the first 10
predictions
pred = model.predict(x_test)
pred[:10]
```

```
array([[0.4989694 ],
       [0.5111768 ],
       [0.4981183 ],
       [0.50972915],
       [0.5059872 ],
       [0.50466985],
       [0.5042962 ],
       [0.5179587 ],
       [0.5002746 ],
       [0.5066786 ]], dtype=float32)
```

0.5

Fail 0 Pass 1 0.5

Pass 1

0.5 0.6

1 DNN

Keras

DL ML

Keras

S2 StatLib

Keras

```
#Download the data using Keras; this will need an active internet connection
from keras.datasets import boston_housing
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

```
#Explore the data structure using basic python commands
print("Type of the Dataset:",type(y_train))
print("Shape of training data :",x_train.shape)
print("Shape of training labels :",y_train.shape)
print("Shape of testing data :",type(x_test))
print("Shape of testing labels :",y_test.shape)
```

```
Type of the Dataset: <class 'numpy.ndarray'>
Shape of training data : (404, 13)
Shape of training labels : (404,)
Shape of testing data : <class 'numpy.ndarray'>
Shape of testing labels : (102,)
```

Python numpy

13 404 Numpy

102

102 300 402 80:20

CMU
http://lib.stat.cmu.edu/datasets/boston

Column Name	Description
CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq. ft.
INDUS	proportion of nonretail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxide concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centers
RAD	index of accessibility to radial highways
TAX	full-value property tax rate per $10,000
PTRATIO	pupil-teacher ratio by town
B	1000(Bk – 0.63)^2, where Bk is the proportion of blacks by town
LSTAT	% lower status of the population
MEDV	median value of owner-occupied homes in $1000's

numpy index-slicing

numpy n

```
x_train[:3,:]

array([[ 1.23247e+00, 0.00000e+00, 8.14000e+00, 0.00000e+00,
5.38000e-01, 6.14200e+00, 9.17000e+01, 3.97690e+00,
4.00000e+00, 3.07000e+02, 2.10000e+01, 3.96900e+02,
1.87200e+01],
[ 2.17700e-02, 8.25000e+01, 2.03000e+00, 0.00000e+00,
4.15000e-01, 7.61000e+00, 1.57000e+01, 6.27000e+00,
2.00000e+00, 3.48000e+02, 1.47000e+01, 3.95380e+02,
3.11000e+00],
[ 4.89822e+00, 0.00000e+00, 1.81000e+01, 0.00000e+00,
```

6.31000e-01, 4.97000e+00, 1.00000e+02, 1.33250e+00,
2.40000e+01, 6.66000e+02, 2.02000e+01, 3.75520e+02,
3.26000e+00]])

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
#Extract the last 100 rows from the training data to create the
validation datasets.
x_val = x_train[300:,]
y_val = y_train[300:,]
#Define the model architecture
model = Sequential()
model.add(Dense(13, input_dim=13, kernel_initializer='normal',activation='relu'))
model.add(Dense(6, kernel_initializer='normal',activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_percentage_error'])
#Train the model
model.fit(x_train, y_train, batch_size=32, epochs=3,validation_data=(x_val,y_val))
```

```
Train on 404 samples, validate on 104 samples
Epoch 1/3
404/404 [==============================] - 2s 4ms/step - loss:
598.8595 - mean_absolute_percentage_error: 101.7889 - val_loss:
681.4912 - val_mean_absolute_percentage_error: 100.0789
Epoch 2/3
404/404 [==============================] - 0s 81us/step - loss:
583.6991 - mean_absolute_percentage_error: 99.7594 - val_loss:
674.8345 - val_mean_absolute_percentage_error: 99.2616
Epoch 3/3
404/404 [==============================] - 0s 94us/step - loss:
573.6101 - mean_absolute_percentage_error: 98.3180 - val_loss:
654.3787 - val_mean_absolute_percentage_error: 96.9662
```

MAPE

%10

```
results = model.evaluate(x_test, y_test)
for i in range(len(model.metrics_names)):
print(model.metrics_names[i]," : ", results[i])
```

Output

```
102/102 [==============================] - 0s 87us/step
loss : 589.7658882889093
mean_absolute_percentage_error : 96.48218611174939
```

%96 MAPE

96

20 10

DL

30 3

1 30

```
#Train the model
model.fit(x_train, y_train, batch_size=32, epochs=30,
validation_data=(x_val,y_val))
```

Output

```
Train on 404 samples, validate on 104 samples
Epoch 1/1000
404/404 [==============================] - 0s 114us/step -
loss: 536.6662 - mean_absolute_percentage_error: 93.4381 - val_
loss: 580.3155 - val_mean_absolute_percentage_error: 88.6968
Epoch 2/1000
404/404 [==============================] - 0s 143us/step -
loss: 431.7025 - mean_absolute_percentage_error: 79.0697 - val_
loss: 413.4064 - val_mean_absolute_percentage_error: 67.0769
```

Skipping the output for in-between epochs.

(Adding output for only the last three epochs, i.e., 28 to 30)

```
Epoch 28/30
404/404 [==============================] - 0s 111us/step -
loss: 6.0758 - mean_absolute_percentage_error: 9.5185 - val_
loss: 5.2524 - val_mean_absolute_percentage_error: 8.3853
```

```
Epoch 29/30
404/404 [==============================] - 0s 100us/step -
loss: 6.2895 - mean_absolute_percentage_error: 10.1037 - val_
loss: 6.0818 - val_mean_absolute_percentage_error: 8.9386
Epoch 30/30
404/404 [==============================] - 0s 111us/step -
loss: 6.0761 - mean_absolute_percentage_error: 9.8201 - val_
loss: 7.3844 - val_mean_absolute_percentage_error: 8.9812
```

96 MAPE 8.9

```
results = model.evaluate(x_test, y_test)
for i in range(len(model.metrics_names)):
print(model.metrics_names[i]," : ", results[i])
```

Output

```
102/102 [==============================] - 0s 92us/step
loss : 22.09559840782016
mean_absolute_percentage_error : 16.22196163850672
```

MAPE

Keras DNN

Keras DL