# Deep Learning with Watson Studio: Time Series Prediction

IBM Developer

Jerome Nilmeier
Data Scientist and Developer Advocate
IBM CODAIT: **C**enter for **O**penSource **D**ata and **AI T**echnologies

nilmeier@us.ibm.com
*@JNilmeier*

# About Me

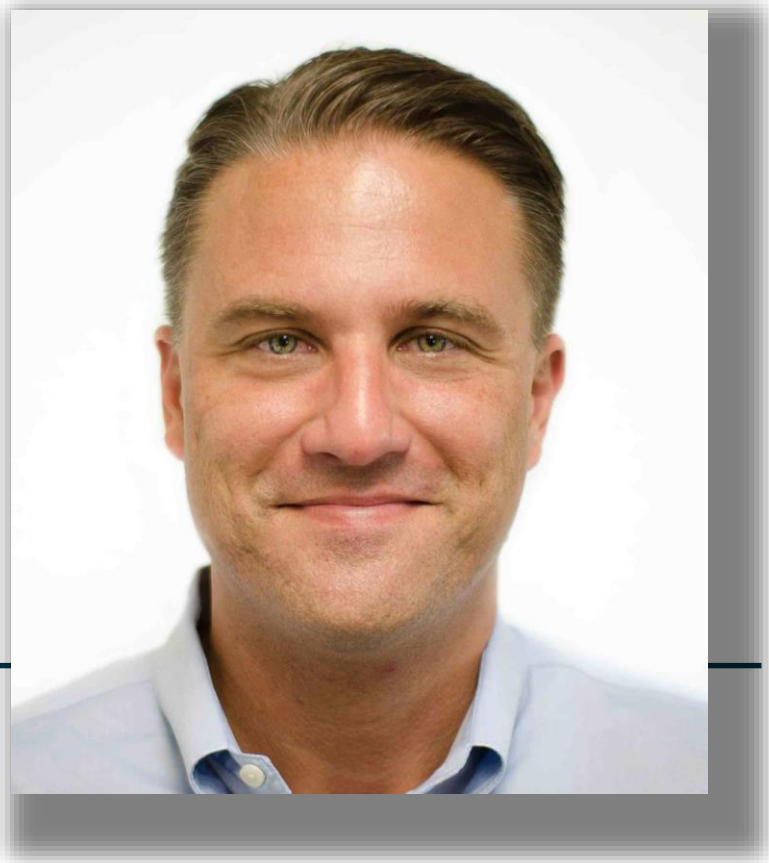
**Jerome Nilmeier**

2015- Present:

Data Scientist and Developer Advocate,

2015-2017:  IBM Spark Technology Center

2017–Present:  IBM CODAIT



**Research Background**

2002: B.S. Chemical Engineering

UC Berkeley

2008: Ph.D. (Computational Biophysics), UC San Francisco

2009-2015:  Postdoctoral Appointments:

- UC Berkeley
- Lawrence Berkeley/Livermore National Labs,
- Stanford OpenMM Fellow
- Insight Data Engineering Fellow

# Open Source Community Leadership

Apache SOFTWARE FOUNDATION

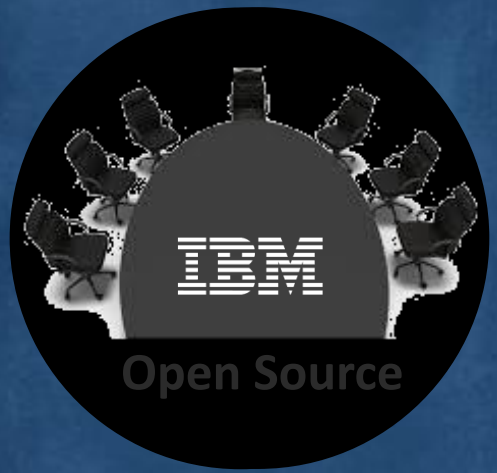
*Founding Partner*

*188+ Project Committers*


*77+ Projects*

amplab UC BERKELEY | riselab UC Berkeley
HYPERLEDGER | R consortium
scalacenter | THE APACHE SOFTWARE FOUNDATION

Key Open source steering committee memberships




OSS Advisory Board

Spark Technology Center

# Center for Open Source Data and AI Technologies

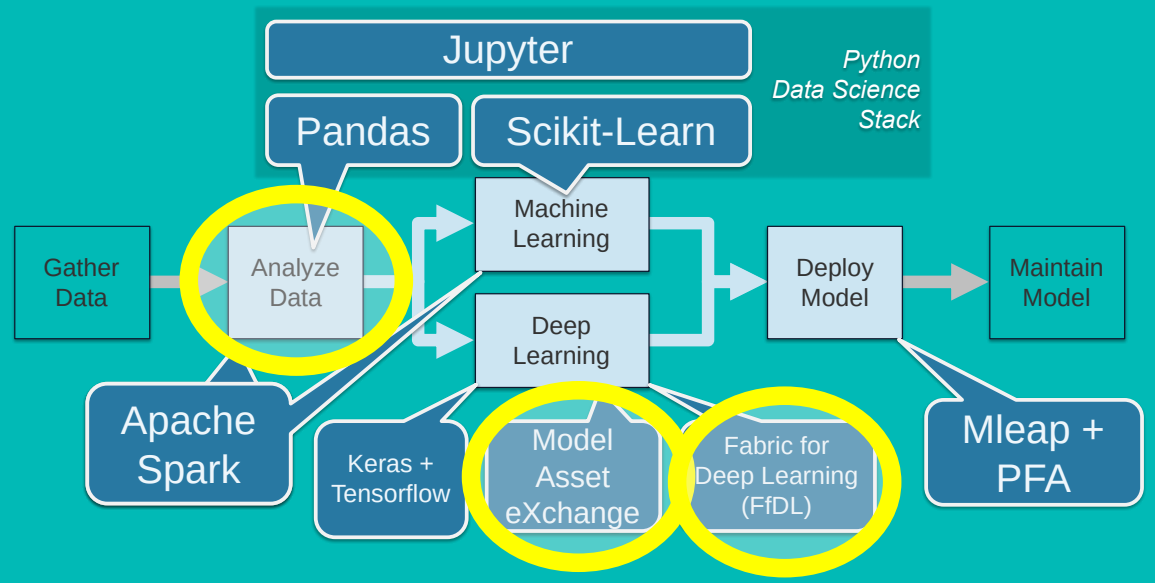
CODAIT aims to make AI solutions dramatically easier to create, deploy, and manage in the enterprise

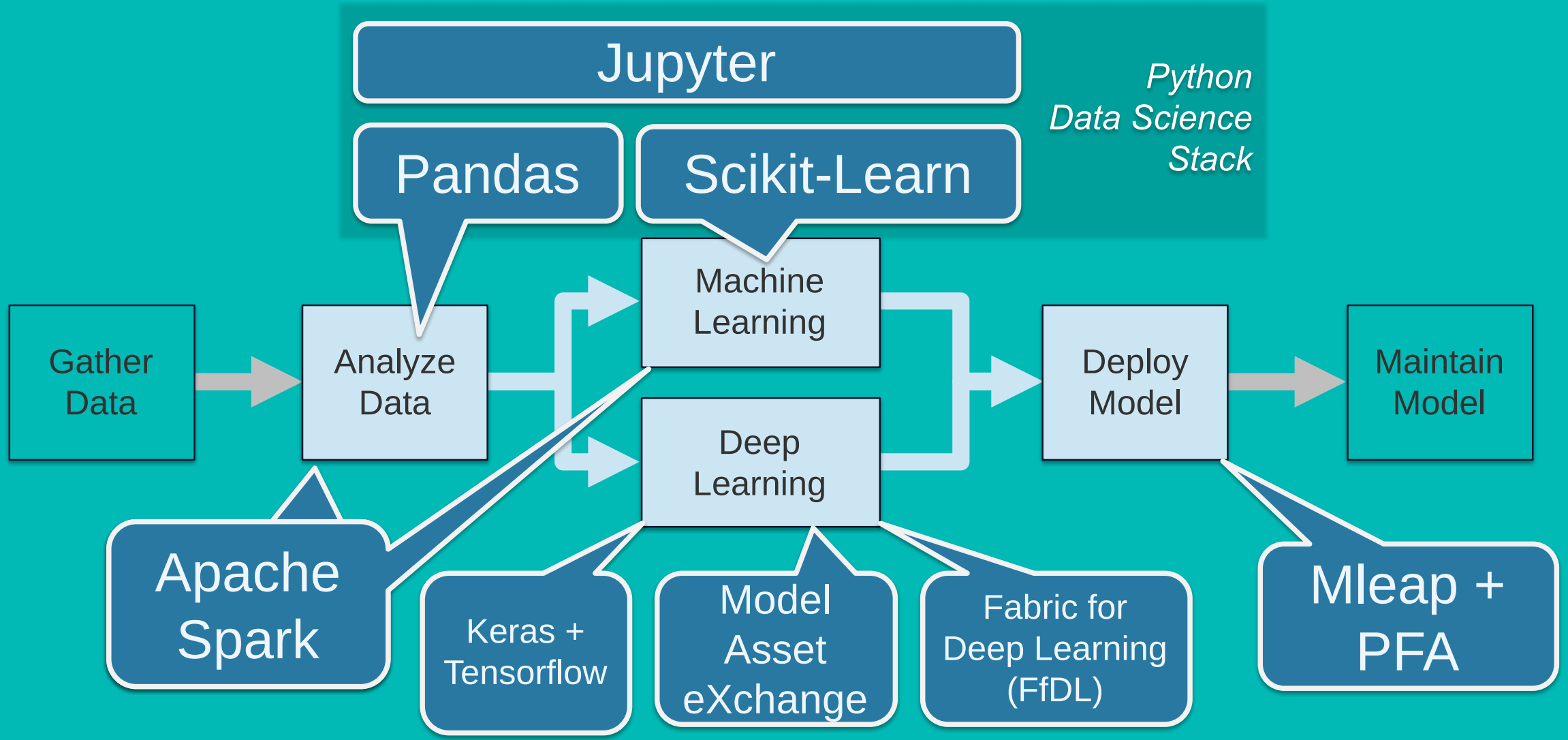
Relaunch of the Spark Technology Center (STC) to reflect expanded mission

CODAIT

codait.org

## Improving Enterprise AI Lifecycle in Open Source




Jupyter
Python
Data Science
Stack
Pandas
Scikit-Learn
Gather
Data
Analyze
Data
Machine
Learning
Deep
Learning
Deploy
Model
Maintain
Model
Apache
Spark
Keras +
Tensorflow
Model
Asset
eXchange
Fabric for
Deep Learning
(FfDL)
Mleap +
PFA

# IBM Developer

# Model Asset eXchange

Free, open-source deep learning models.

Wide variety of domains.

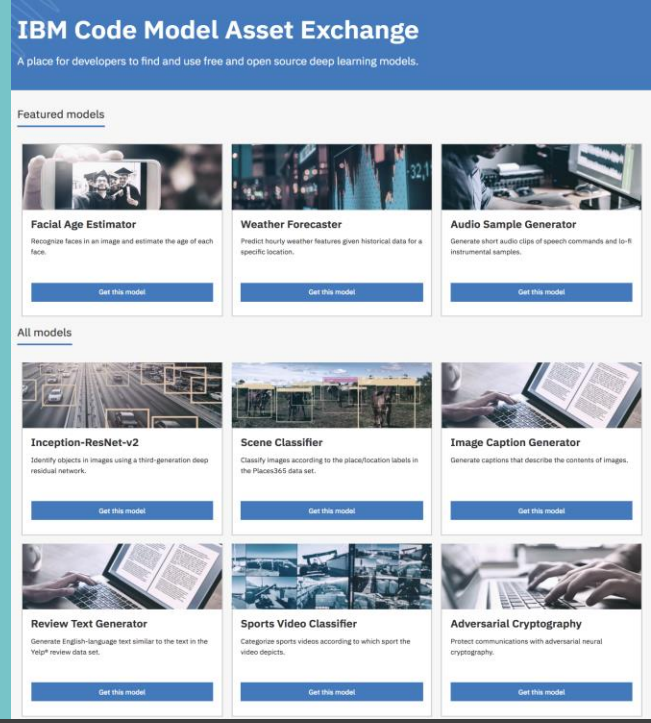
Multiple deep learning frameworks.

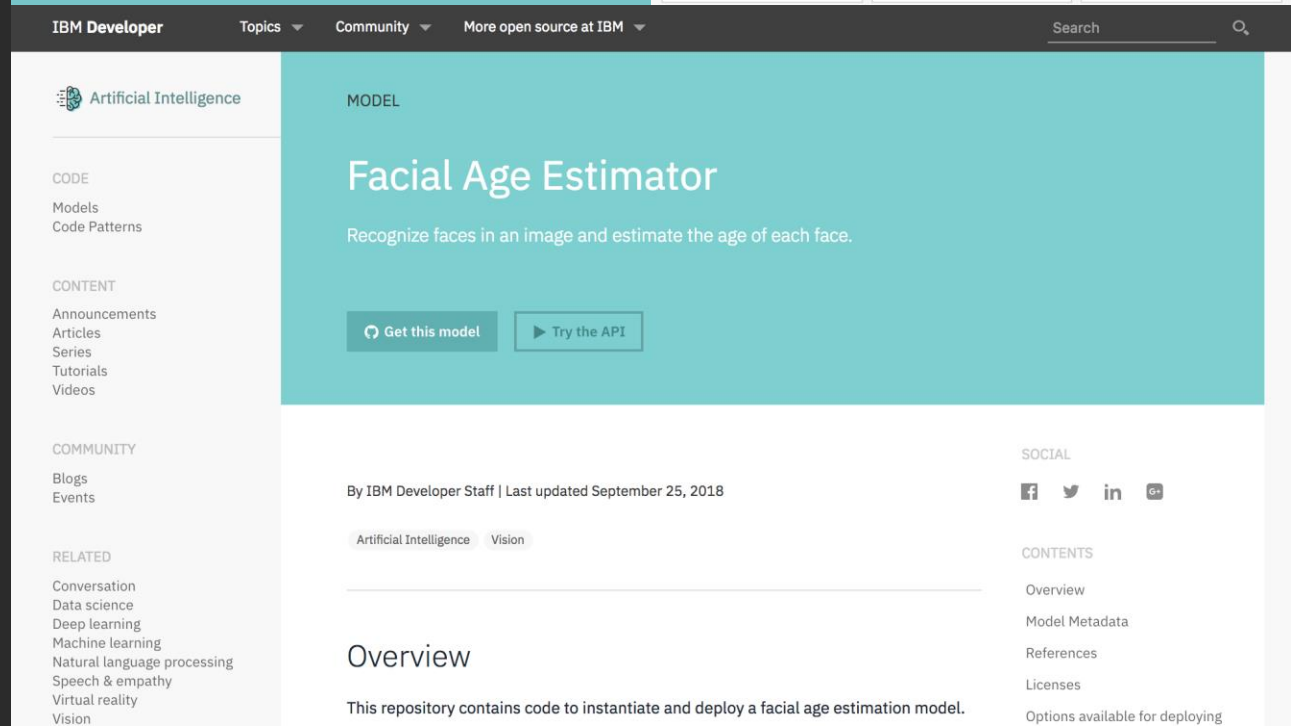
Vetted and tested code and IP.

Build and deploy a web service in 30 seconds.

Start training on **Fabric for Deep Learning (FfDL)** or **Watson Machine Learning** in minutes.

http://ibm.biz/model-exchange



# Summary

## Current status

- 22 models (4 trainable)
- Image, audio, text, healthcare, time-series and more
- 3 Code Patterns demonstrating how to consume MAX models in a web app
- Code Pattern on training an audio classifier using Watson Machine Learning
- One-line deployment via Docker and on a Kubernetes cluster

## Potential Future

- New MAX web portal launching soon
- More deployable models – breadth and depth
- More trainable models - transfer learning in particular
- More MAX-related content:
  - Code Patterns
  - Conference talks, meetups
  - Workshops
- Enhance production-readiness of MAX models
- Improve MAX API framework

# Fabric for Deep Learning

https://github.com/IBM/FfDL

## FfDL provides a scalable, resilient, and fault tolerant deep-learning framework

- Fabric for Deep Learning or FfDL (pronounced as ‘fiddle’) is an open source project which aims at making Deep Learning easily accessible to the people it matters the most i.e. Data Scientists, and AI developers.
- FfDL provides a consistent way to deploy, train and visualize Deep Learning jobs across multiple frameworks like TensorFlow, Caffe, PyTorch, Keras etc.
- FfDL is being developed in close collaboration with IBM Research and IBM Watson. It forms the core of Watson`s Deep Learning service in open source.

FfDL

FfDL Github Page
https://github.com/IBM/FfDL

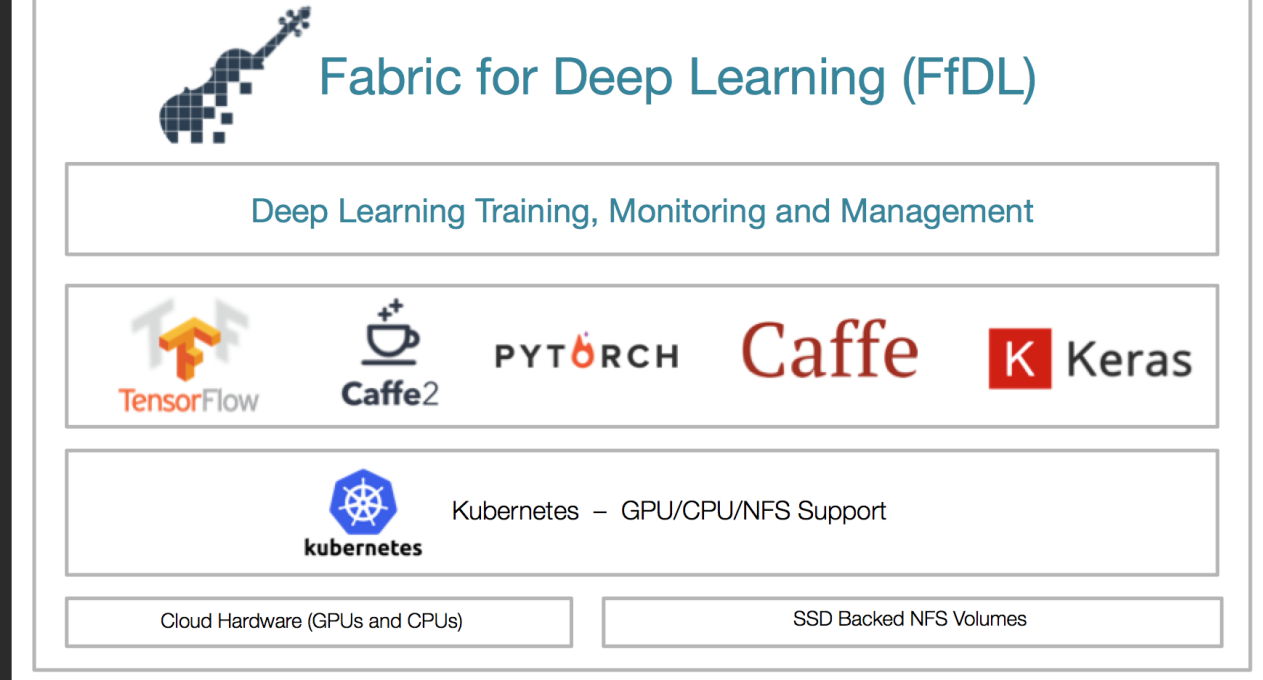
FfDL dwOpen Page
https://developer.ibm.com/code/open/projects/fabric-for-deep-learning-ffdl/

FfDL Announcement Blog
http://developer.ibm.com/code/2018/03/20/fabric-for-deep-learning

FfDL Technical Architecture Blog
http://developer.ibm.com/code/2018/03/20/democratize-ai-with-fabric-for-deep-learning

Deep Learning as a Service within Watson Studio
https://www.ibm.com/cloud/deep-learning

Research paper: “Scalable Multi-Framework Management of Deep Learning Training Jobs”
http://learningsys.org/nips17/assets/papers/paper_29.pdf

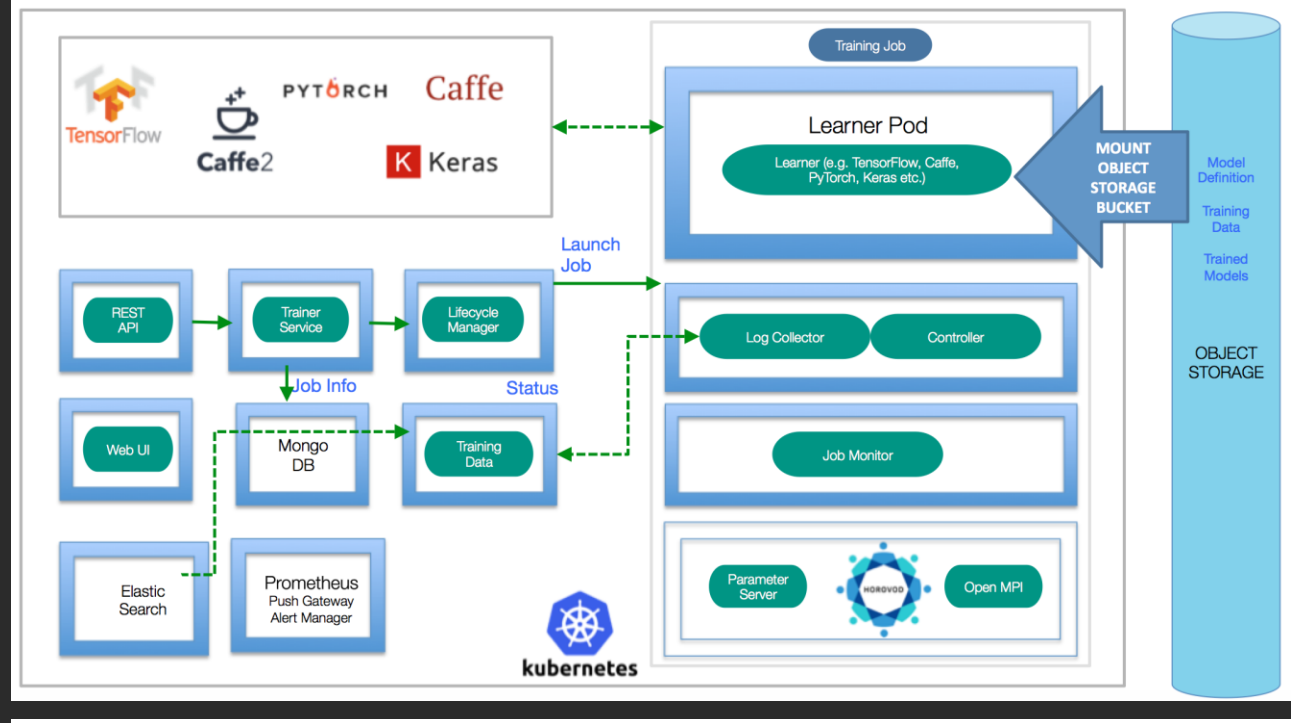


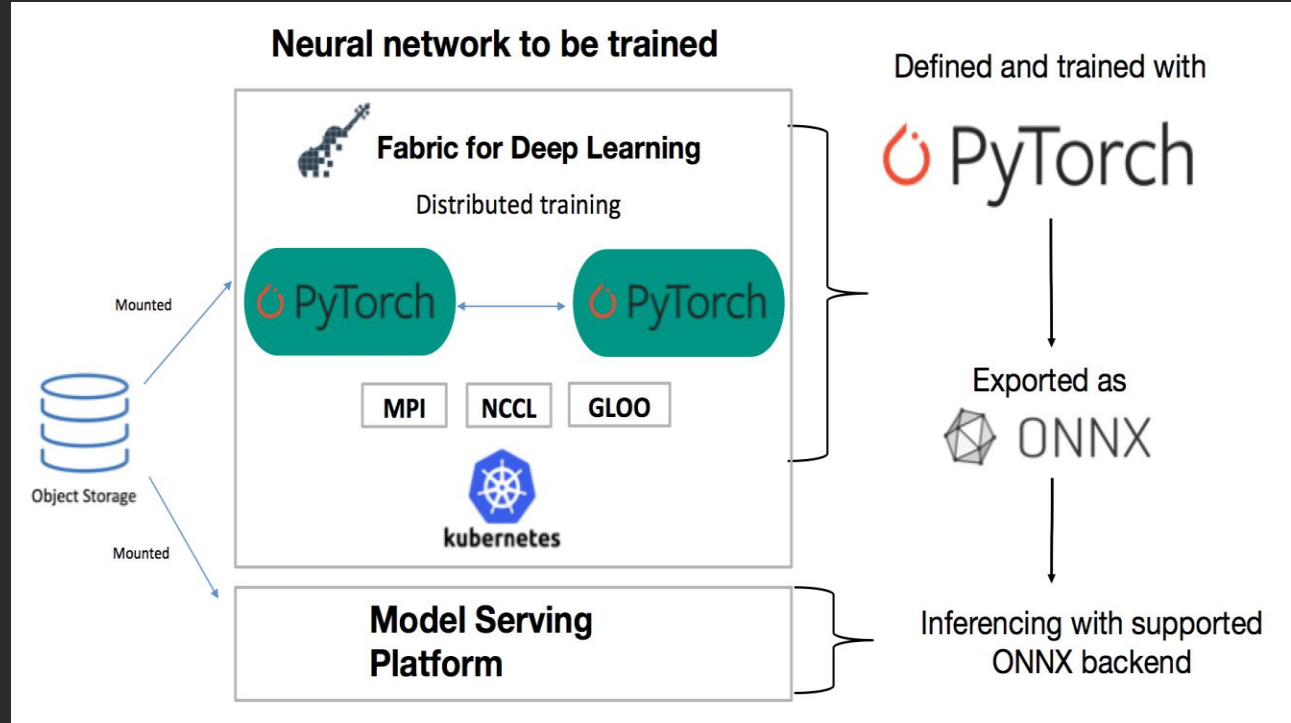
# Fabric for Deep Learning

https://github.com/IBM/FfDL

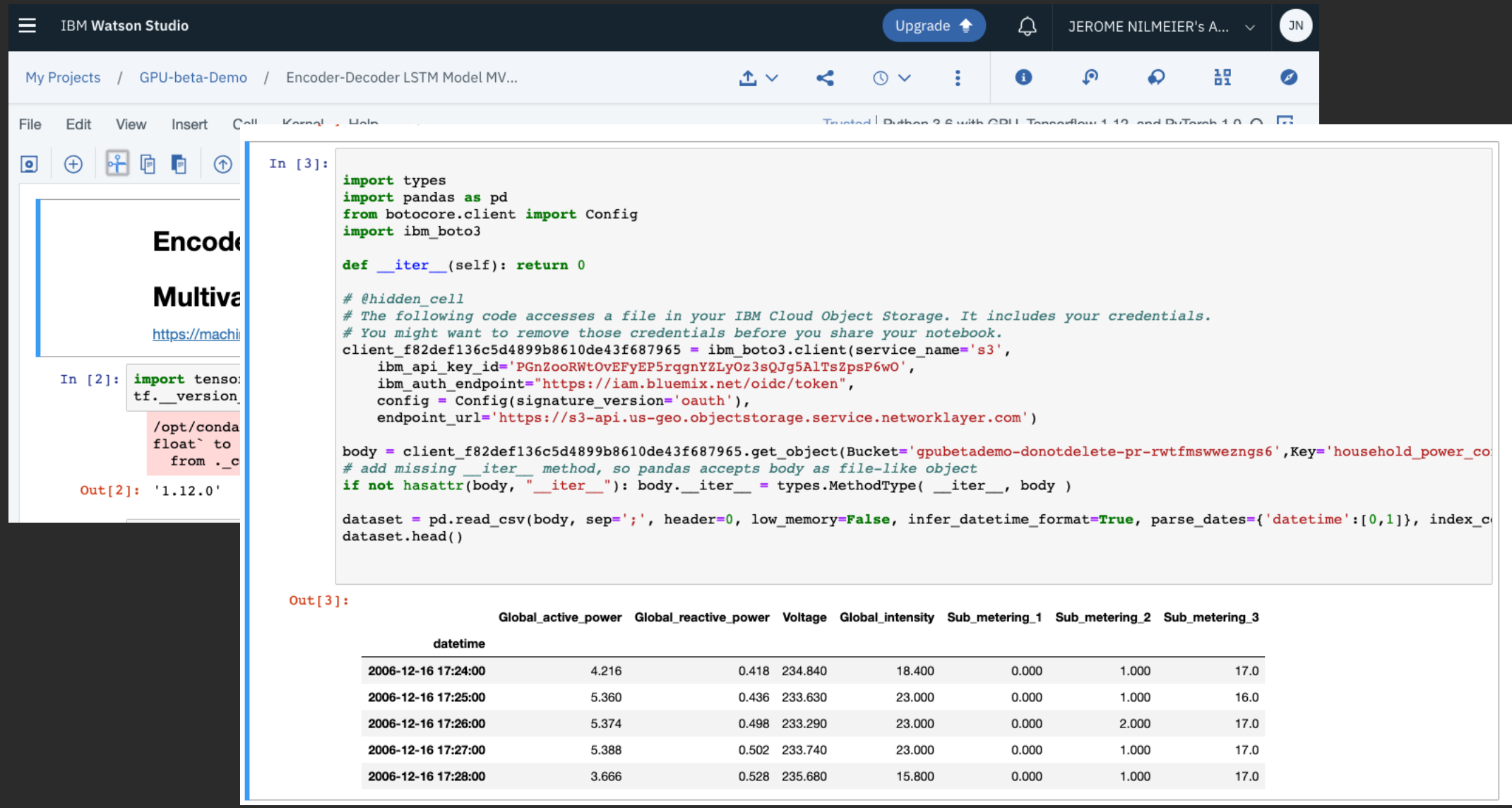
Just announced: Support for PyTorch 1.0 – including distributed training and ONNX!

Supports distributed training via Horovod

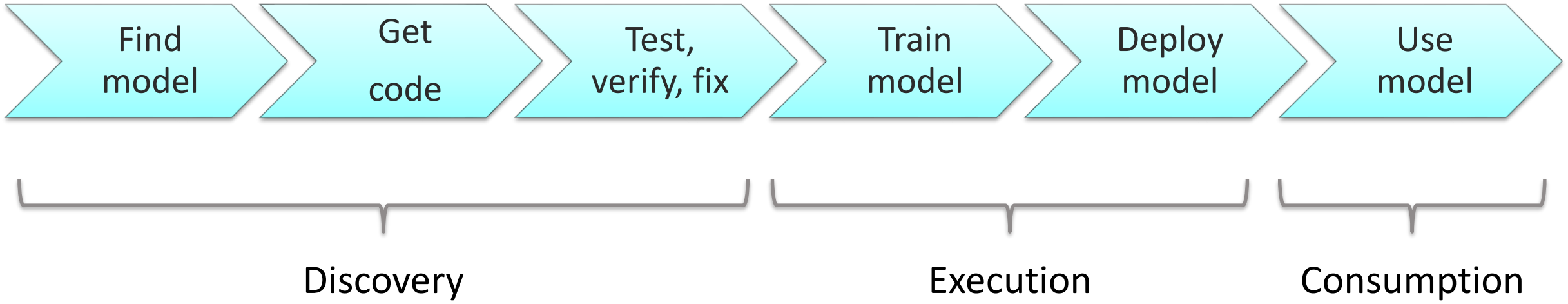


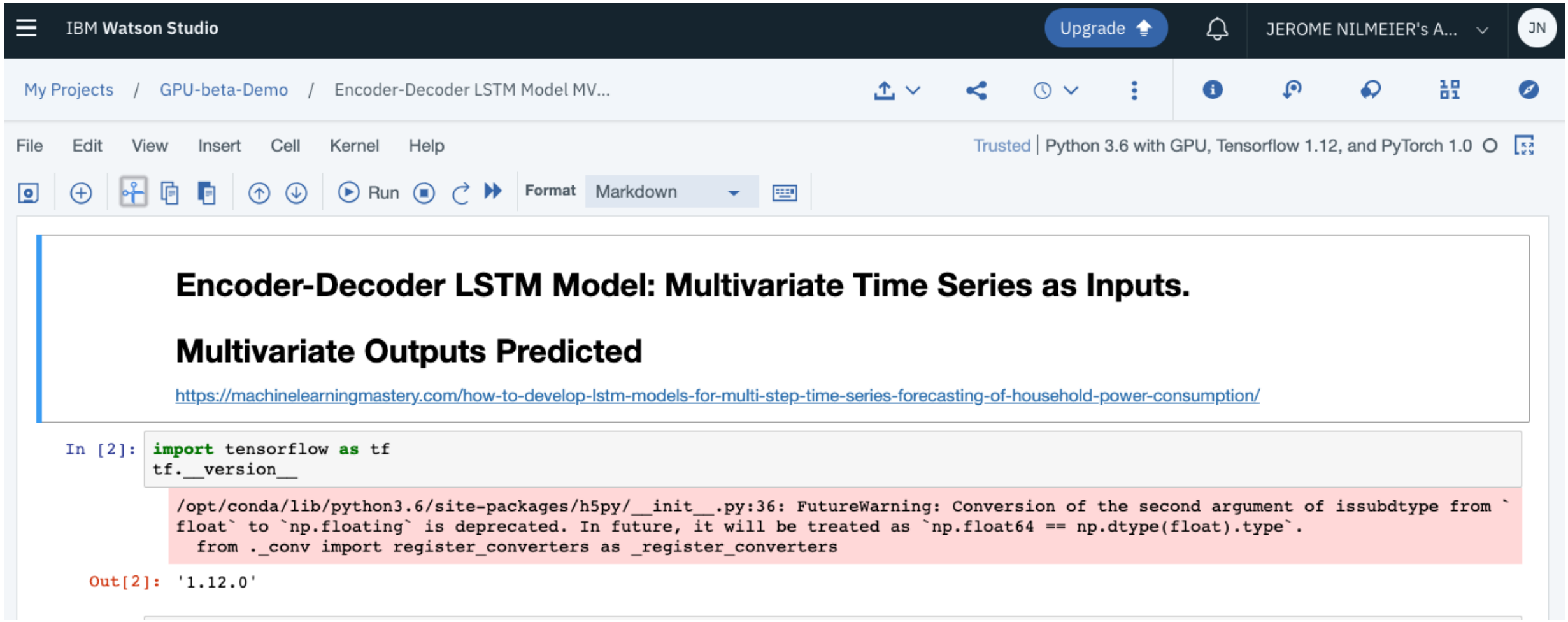
# Discovery


IBM Watson Studio
Upgrade
JERONE NILMEIER's A...
JN
My Projects / GPU-beta-Demo / Encoder-Decoder LSTM Model MV...
File Edit View Insert Cell Kernel Help
Encode
Multiva
https://machi
In [2]:
import tenso
tf.__version_
/opt/conda
float` to
from ._c
Out[2]: '1.12.0'
In [3]:
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3
def __iter__(self): return 0
# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_f82def136c5d4899b8610de43f687965 = ibm_boto3.client(service_name='s3',
ibm_api_key_id='PGnZooRWtOvEFyEP5rqgnYZLyOz3sQJg5AlTsZpsP6wO',
ibm_auth_endpoint="https://iam.bluemix.net/oidc/token",
config = Config(signature_version='oauth'),
endpoint_url='https://s3-api.us-geo.objectstorage.service.networklayer.com')
body = client_f82def136c5d4899b8610de43f687965.get_object(Bucket='gpubetademo-donotdelete-pr-rwtfmswweznqs6',Key='household_power_co
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )
dataset = pd.read_csv(body, sep=';', header=0, low_memory=False, infer_datetime_format=True, parse_dates={'datetime':[0,1]}, index_c
dataset.head()
Out[3]:
Global_active_power Global_reactive_power Voltage Global_intensity Sub_metering_1 Sub_metering_2 Sub_metering_3
datetime
2006-12-16 17:24:00 4.216 0.418 234.840 18.400 0.000 1.000 17.0
2006-12-16 17:25:00 5.360 0.436 233.630 23.000 0.000 1.000 16.0
2006-12-16 17:26:00 5.374 0.498 233.290 23.000 0.000 2.000 17.0
2006-12-16 17:27:00 5.388 0.502 233.740 23.000 0.000 1.000 17.0
2006-12-16 17:28:00 3.666 0.528 235.680 15.800 0.000 1.000 17.0

# Applying Deep Learning: Exploratory Modeling is Still an Important Part of the Process



# Applying Deep Learning: Exploratory Modeling is Still an Important Part of the Process



# Applying Deep Learning: Exploratory Modeling is Still an Important Part of the Process

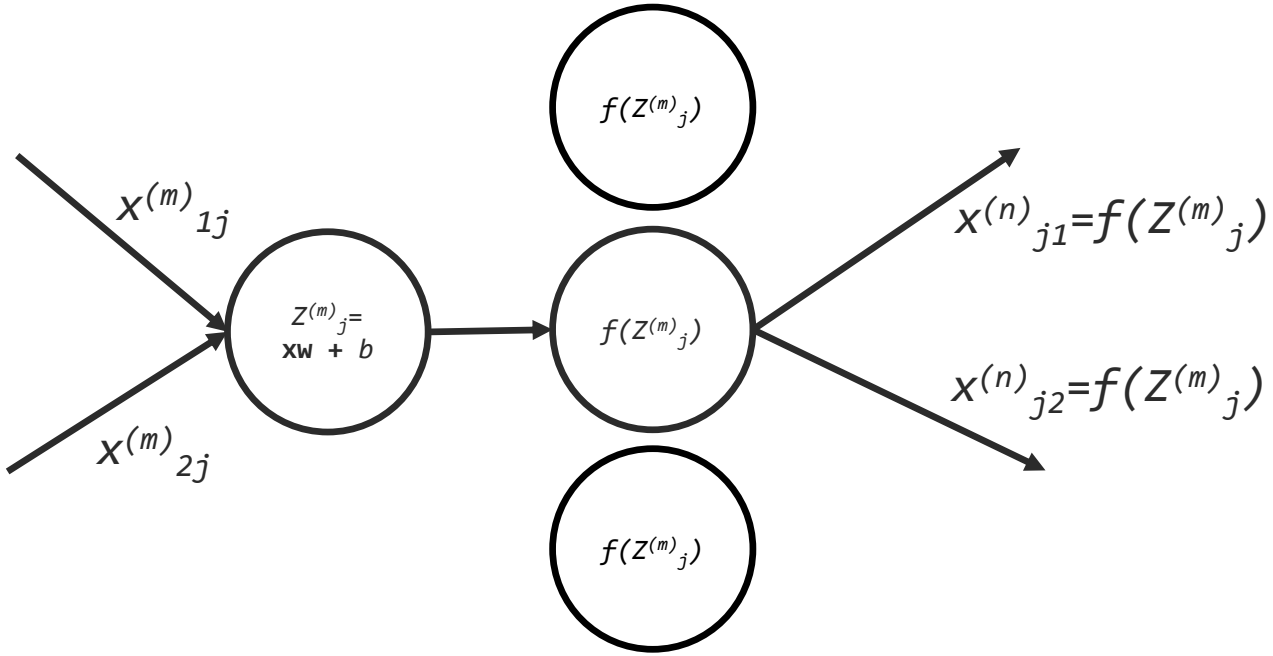
In [3]:

```
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_f82def136c5d4899b8610de43f687965 = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='PGnZooRWtOvEFyEP5rqgnYZLyOz3sQJg5AlTsZpsP6wO',
    ibm_auth_endpoint="https://iam.bluemix.net/oidc/token",
    config = Config(signature_version='oauth'),
    endpoint_url='https://s3-api.us-geo.objectstorage.service.networklayer.com')

body = client_f82def136c5d4899b8610de43f687965.get_object(Bucket='gpubetademo-donotdelete-pr-rwtfmswweznqs6',Key='household_power_co
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

dataset = pd.read_csv(body, sep=';', header=0, low_memory=False, infer_datetime_format=True, parse_dates={'datetime':[0,1]}, index_c
dataset.head()
```

Out[3]:

| datetime | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 |
|---|---|---|---|---|---|---|---|
| 2006-12-16 17:24:00 | 4.216 | 0.418 | 234.840 | 18.400 | 0.000 | 1.000 | 17.0 |
| 2006-12-16 17:25:00 | 5.360 | 0.436 | 233.630 | 23.000 | 0.000 | 1.000 | 16.0 |
| 2006-12-16 17:26:00 | 5.374 | 0.498 | 233.290 | 23.000 | 0.000 | 2.000 | 17.0 |
| 2006-12-16 17:27:00 | 5.388 | 0.502 | 233.740 | 23.000 | 0.000 | 1.000 | 17.0 |
| 2006-12-16 17:28:00 | 3.666 | 0.528 | 235.680 | 15.800 | 0.000 | 1.000 | 17.0 |

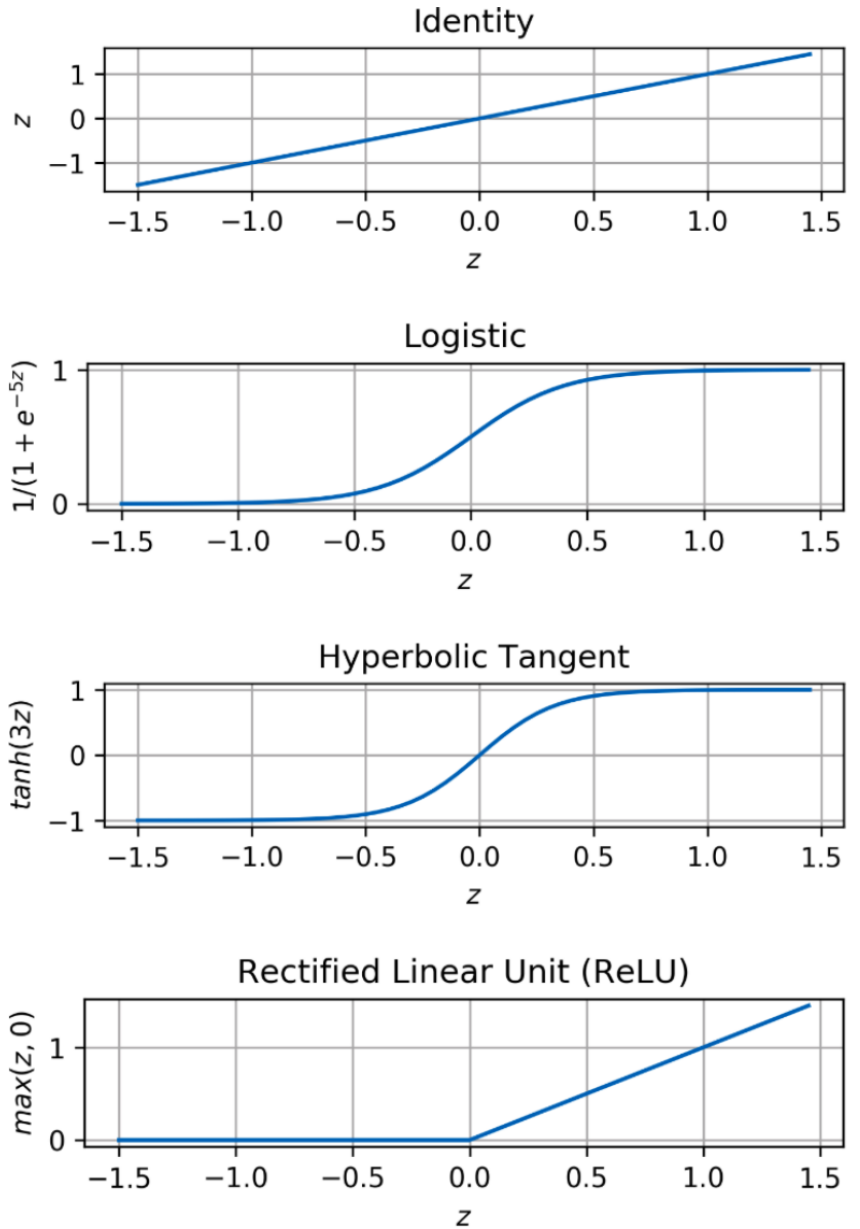
# Neurons:
# The Basic Unit of a Neural Network



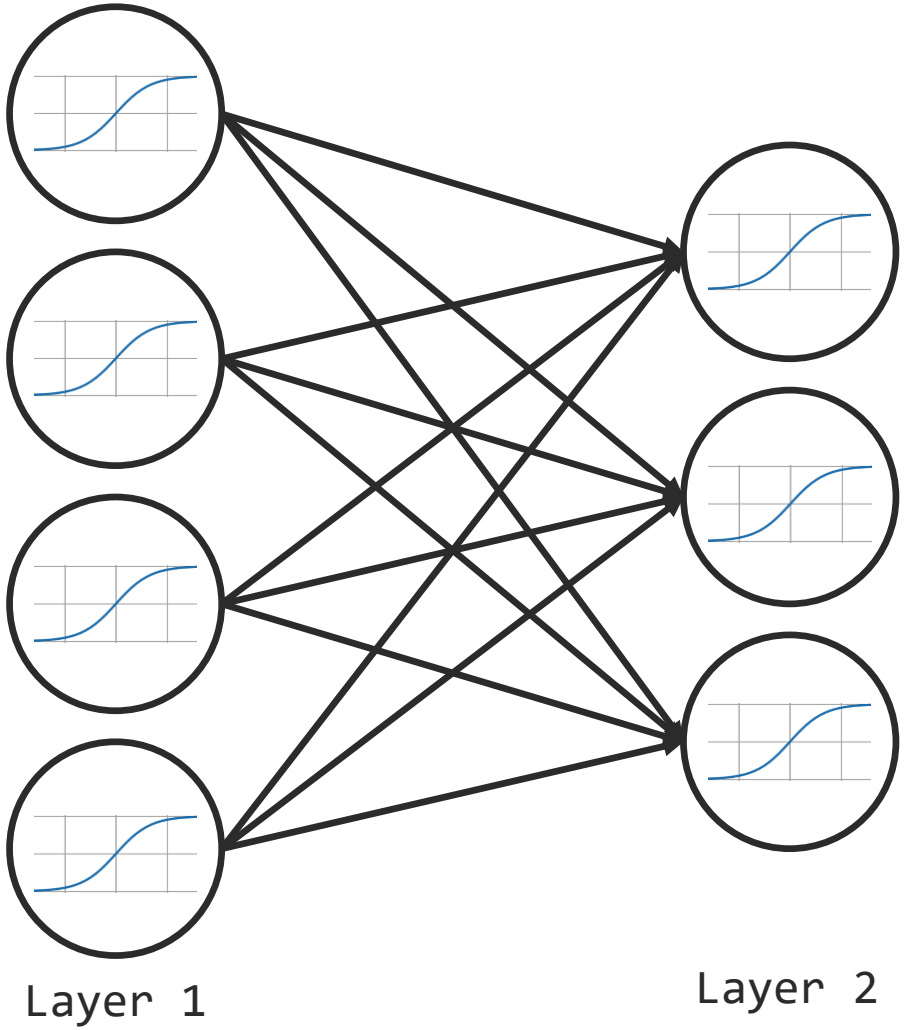

Layer *n*

$f(Z^{(m)}_j)$ *is the* ***Activation Function***

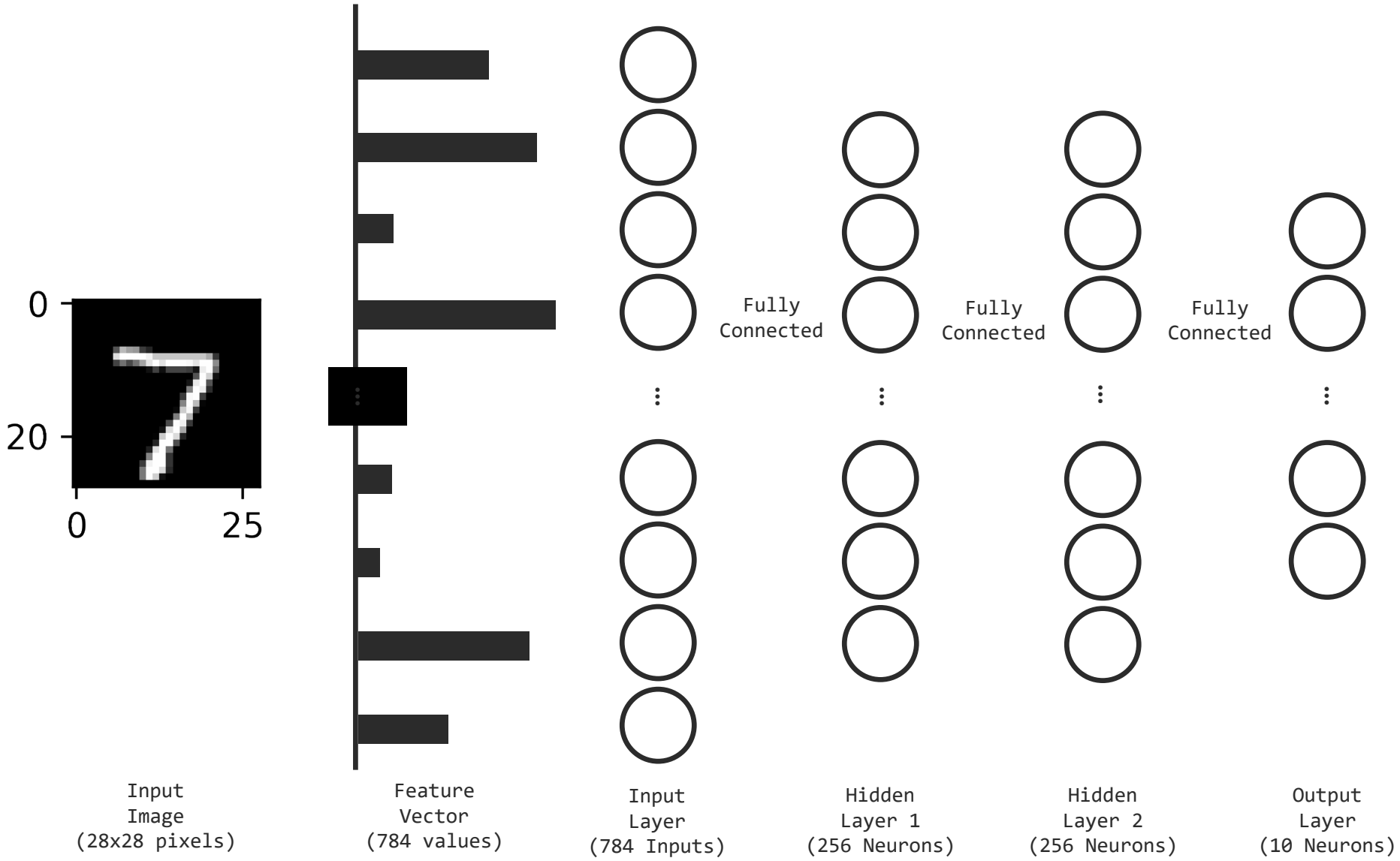
***Activation Functions***



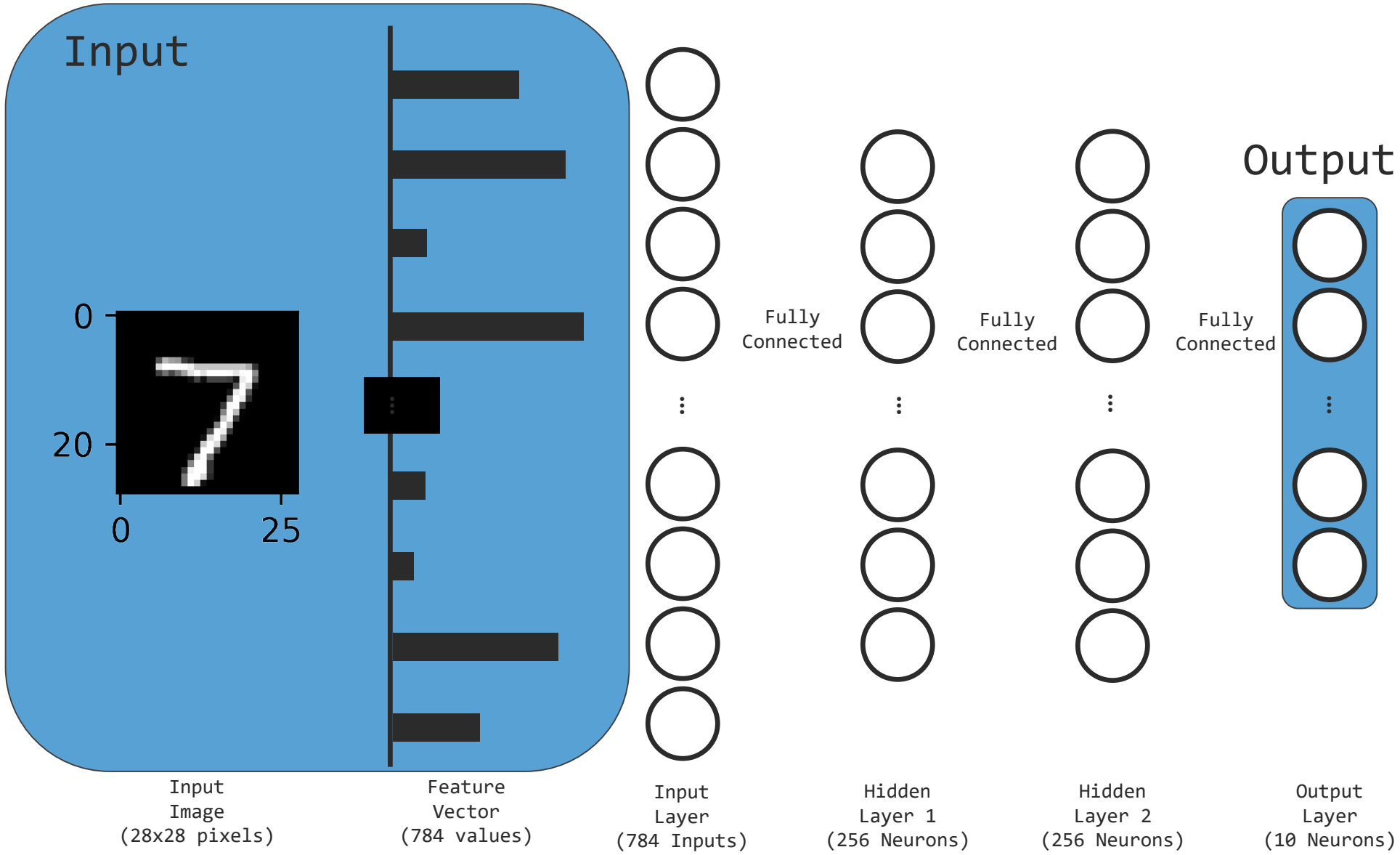
# Neural Network Layers: The Dense Layer



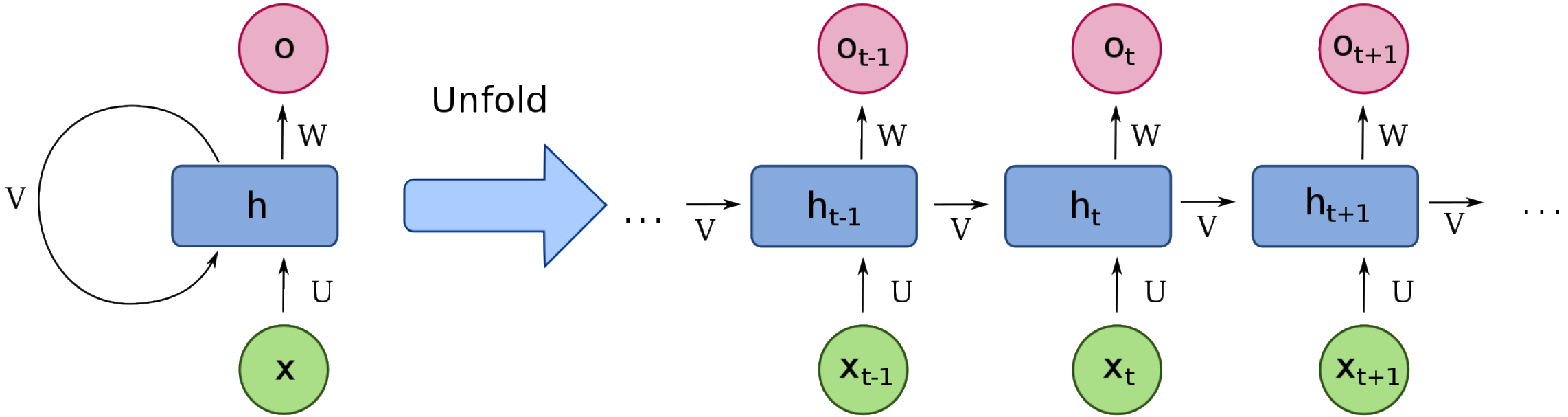
# A Simple Neural Network for Identifying Numbers



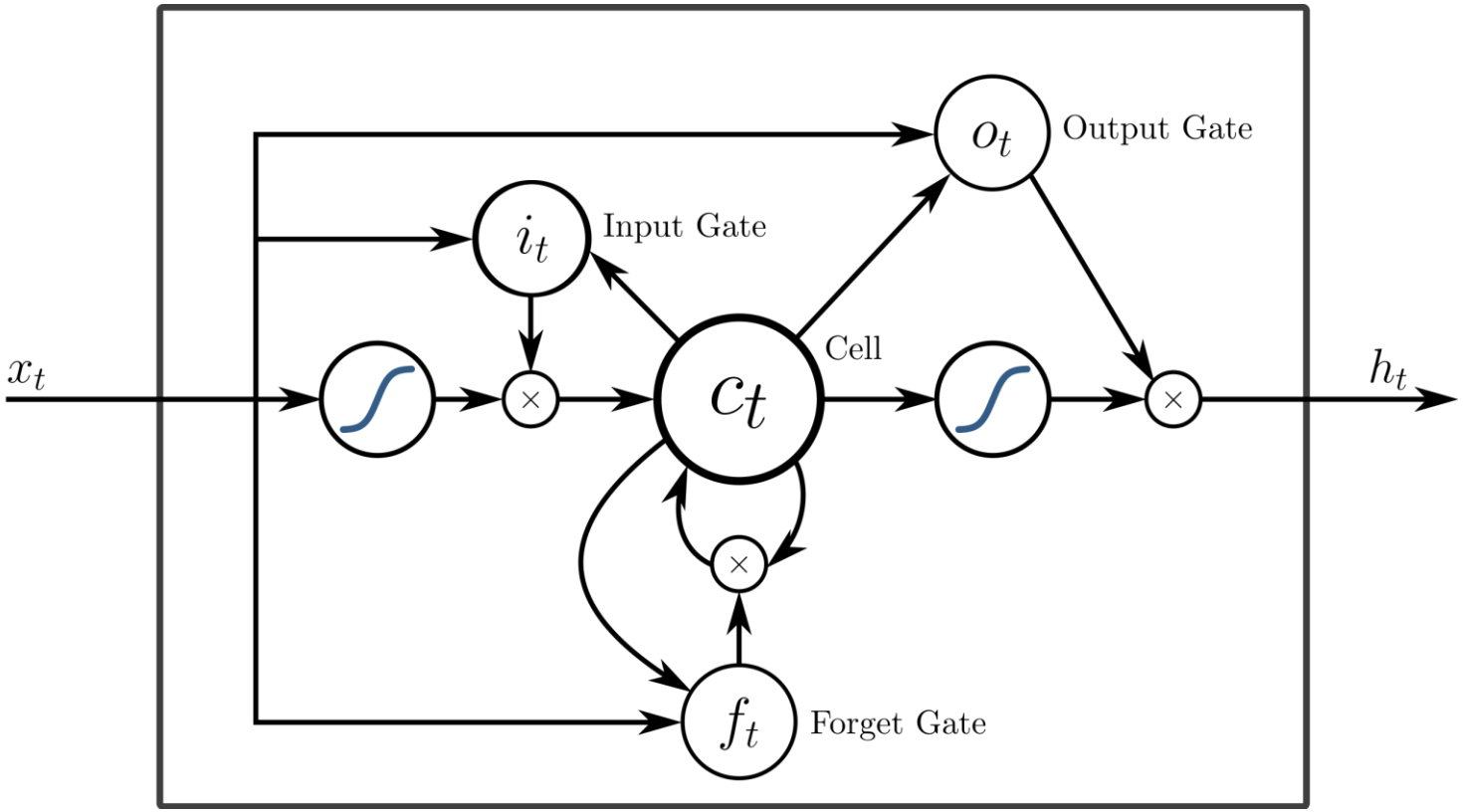
# A Simple Neural Network for Identifying Numbers



# Recurrent Neural Networks



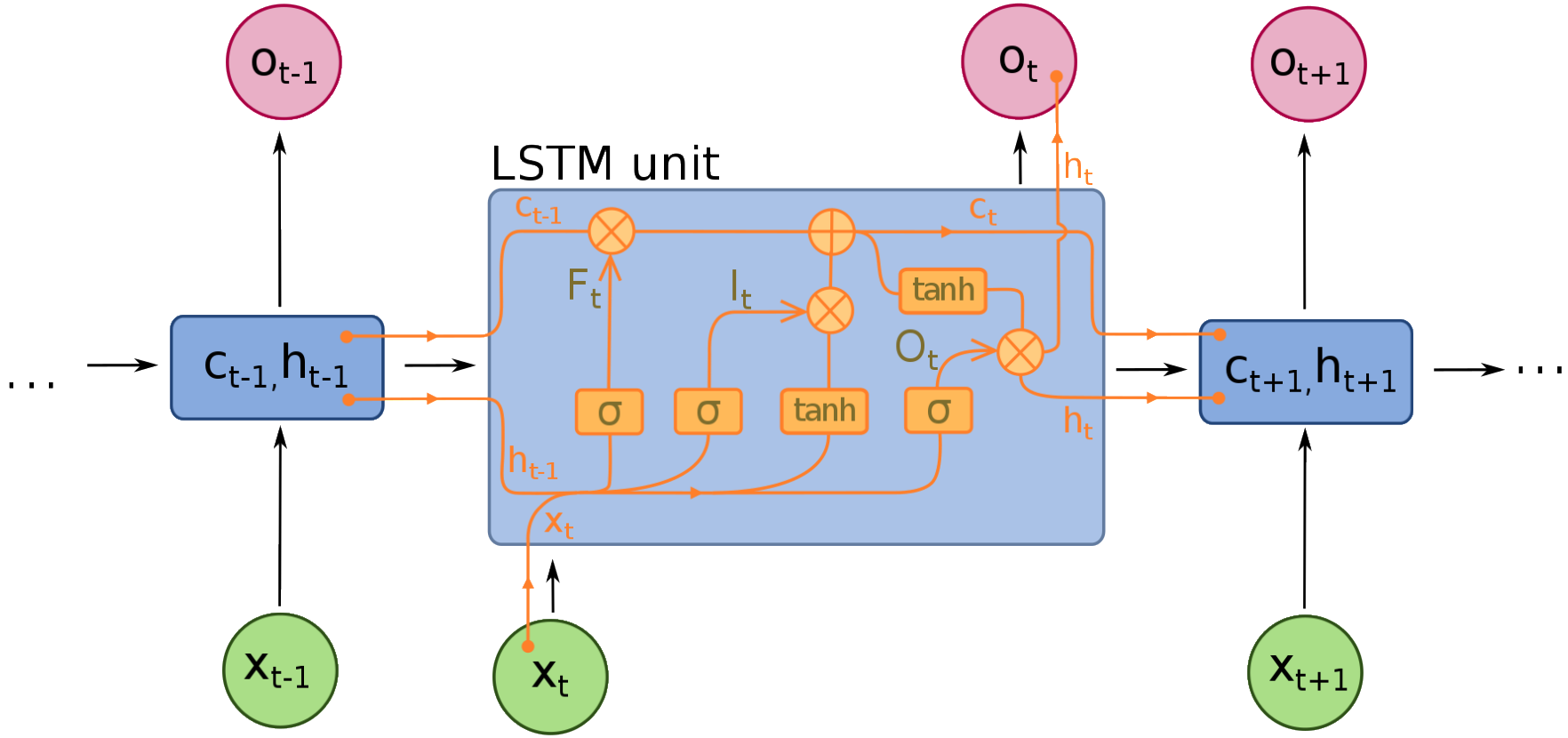
# The LSTM Neuron:
# Long Short Term Memory




https://en.wikipedia.org/wiki/Long_short-term_memory

https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:Peephole_Long_Short-Term_Memory.svg

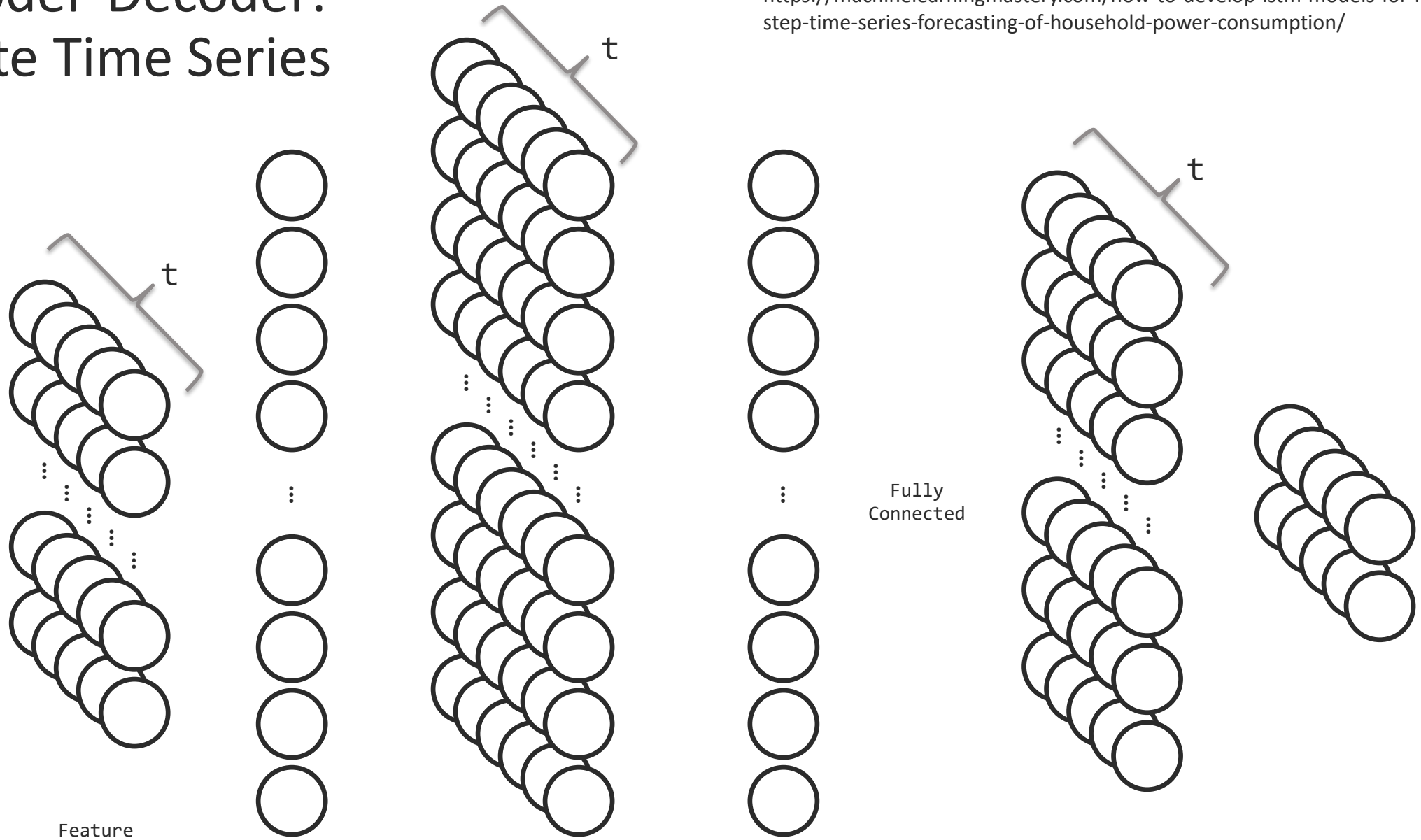
# Recurrent Neural Networks



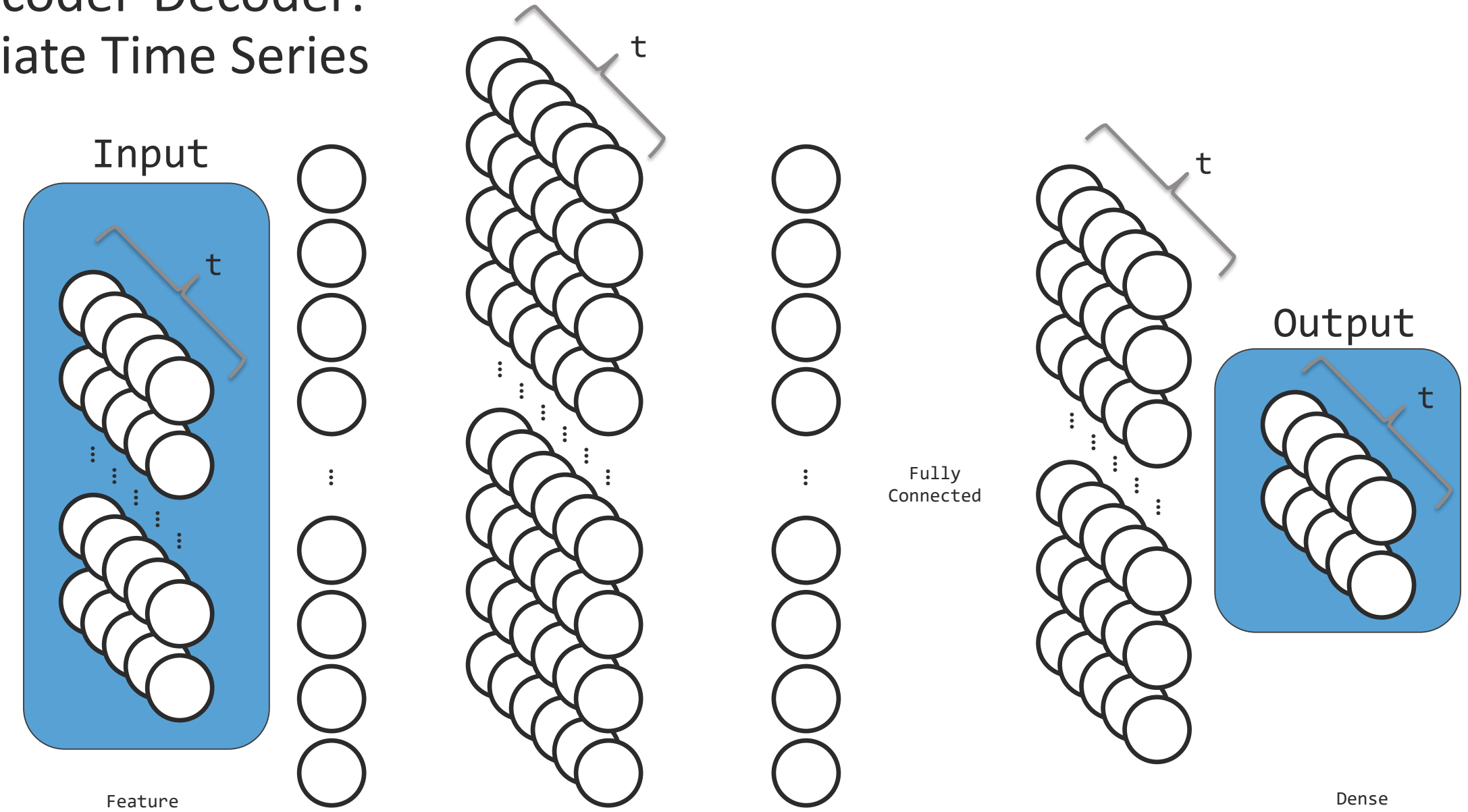

https://en.wikipedia.org/wiki/Recurrent_neural_network

# LSTM Encoder-Decoder: Multivariate Time Series

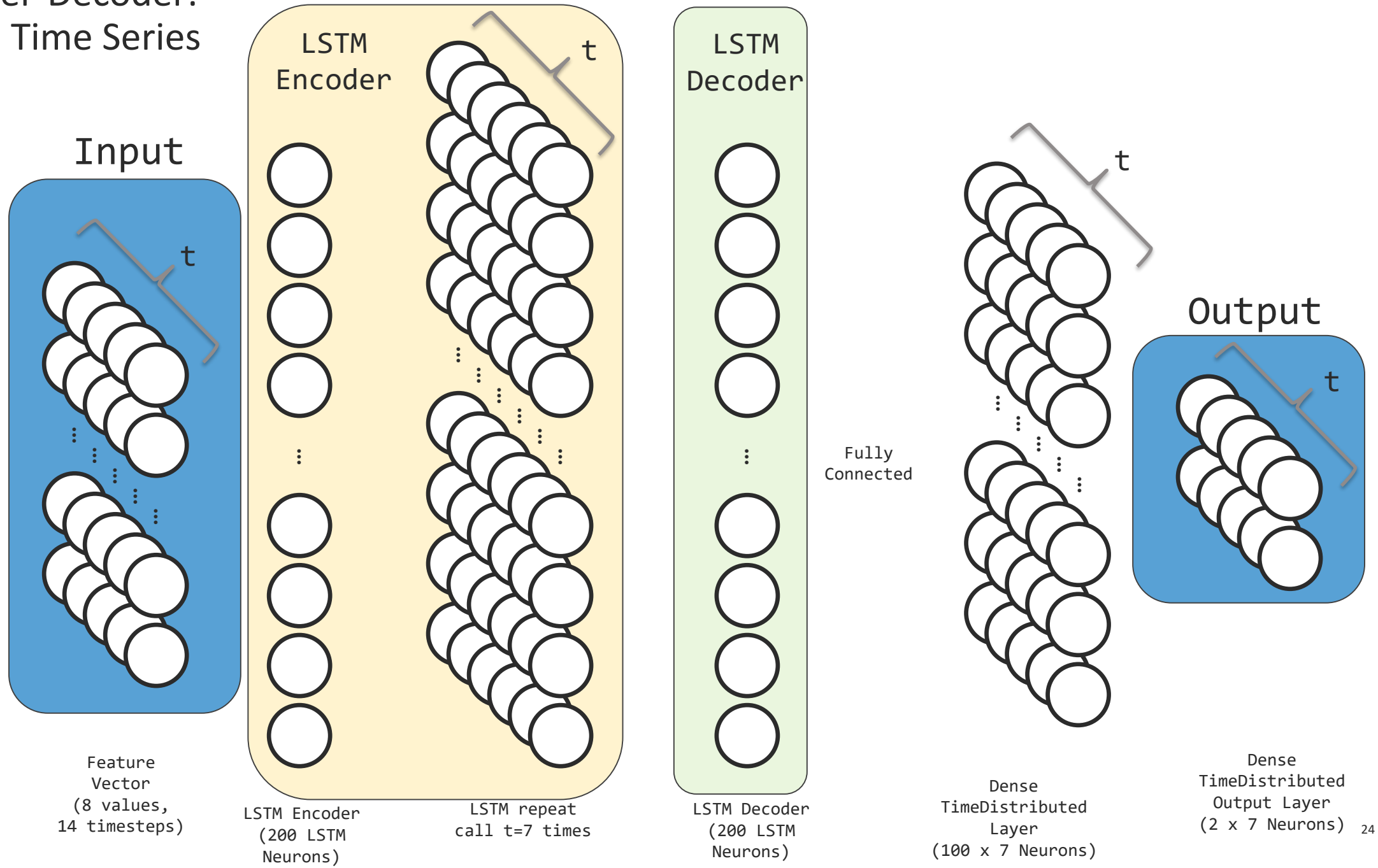
Modified from https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/



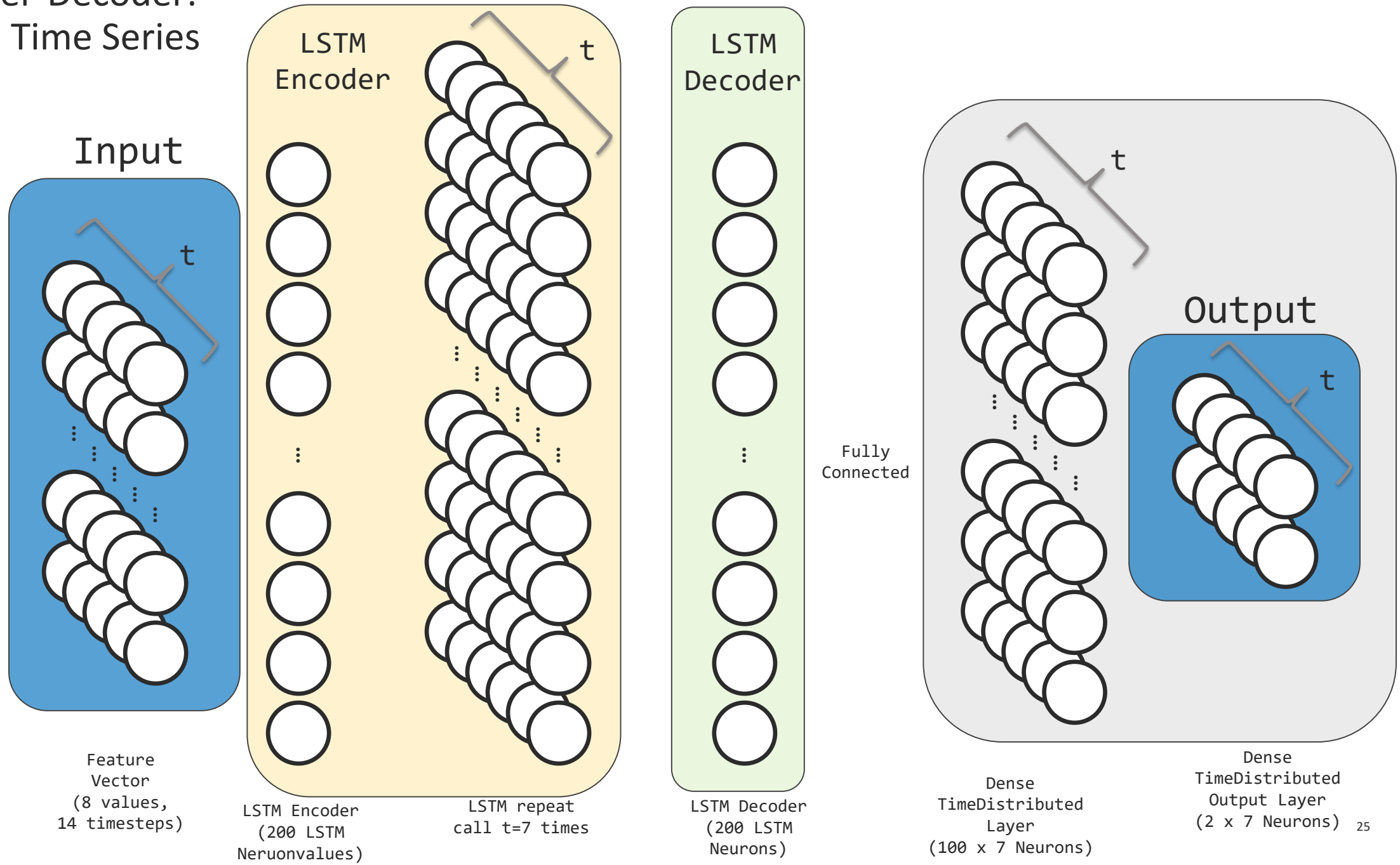
# LSTM Encoder-Decoder: Multivariate Time Series



# LSTM Encoder-Decoder: Multivariate Time Series



# LSTM Encoder-Decoder: Multivariate Time Series



# Using Watson Studio For GPU Accelerated Notebooks

# Using Keras for Network Modeling

In [24]:
```
verbose, epochs, batch_size = 1, 50, 16
n_timesteps, n_features, n_outputs = train_x.shape[1], train_x.shape[2], train_y.shape[1]
# reshape output into [samples, timesteps, features]
train_y = train_y.reshape((train_y.shape[0], train_y.shape[1], train_y.shape[2]))
# define model
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_timesteps, n_features)))
model.add(RepeatVector(n_outputs))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(100, activation='relu')))
model.add(TimeDistributed(Dense(2)))
model.compile(loss='mse', optimizer='adam')
# fit network
model.fit(train_x, train_y, epochs=epochs, batch_size=batch_size, verbose=verbose)
```

1092 Examples in Training Set

```
Epoch 1/50
1092/1092 [==============================] - 7s 6ms/step - loss: 45696059.2821
Epoch 2/50
1092/1092 [==============================] - 2s 2ms/step - loss: 32924188.2363

                                          ⋮

Epoch 49/50
1092/1092 [==============================] - 2s 2ms/step - loss: 133382.4874
Epoch 50/50
1092/1092 [==============================] - 2s 2ms/step - loss: 133650.8616
```

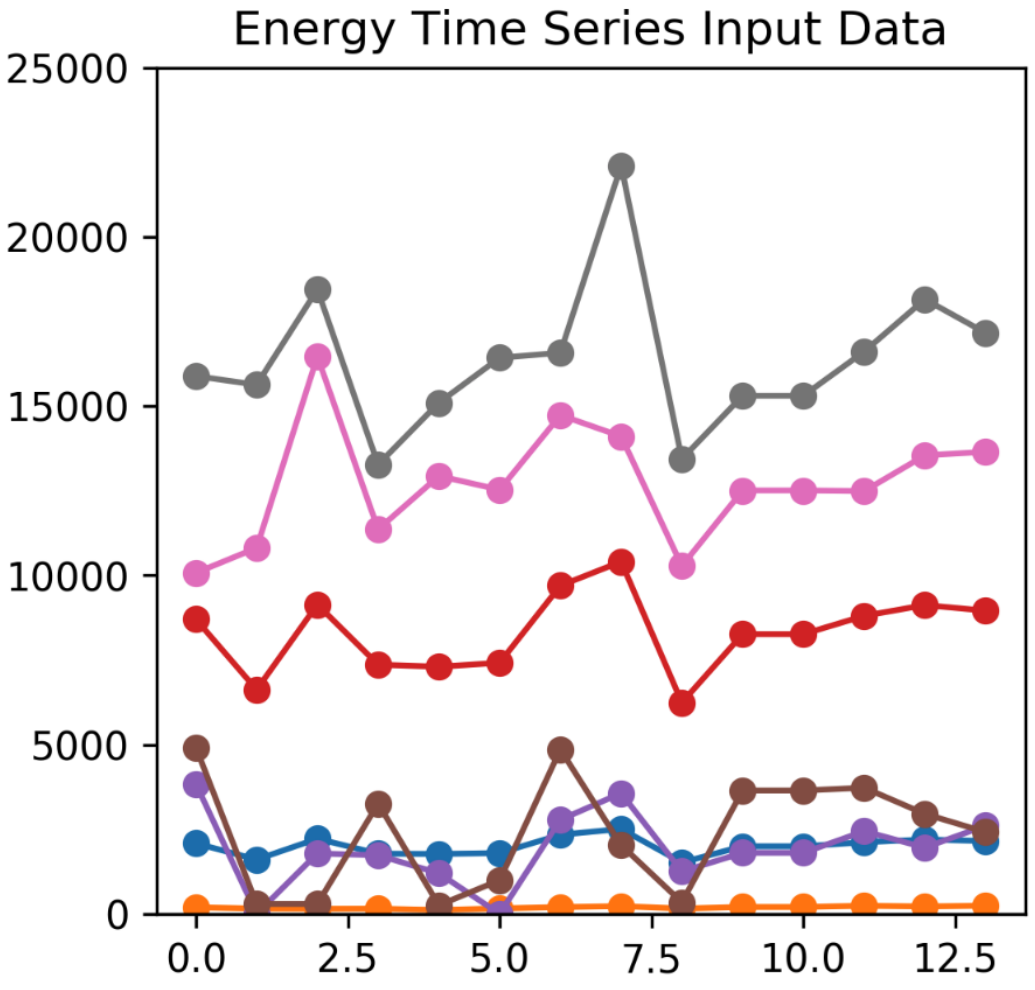
```
model.summary()
```

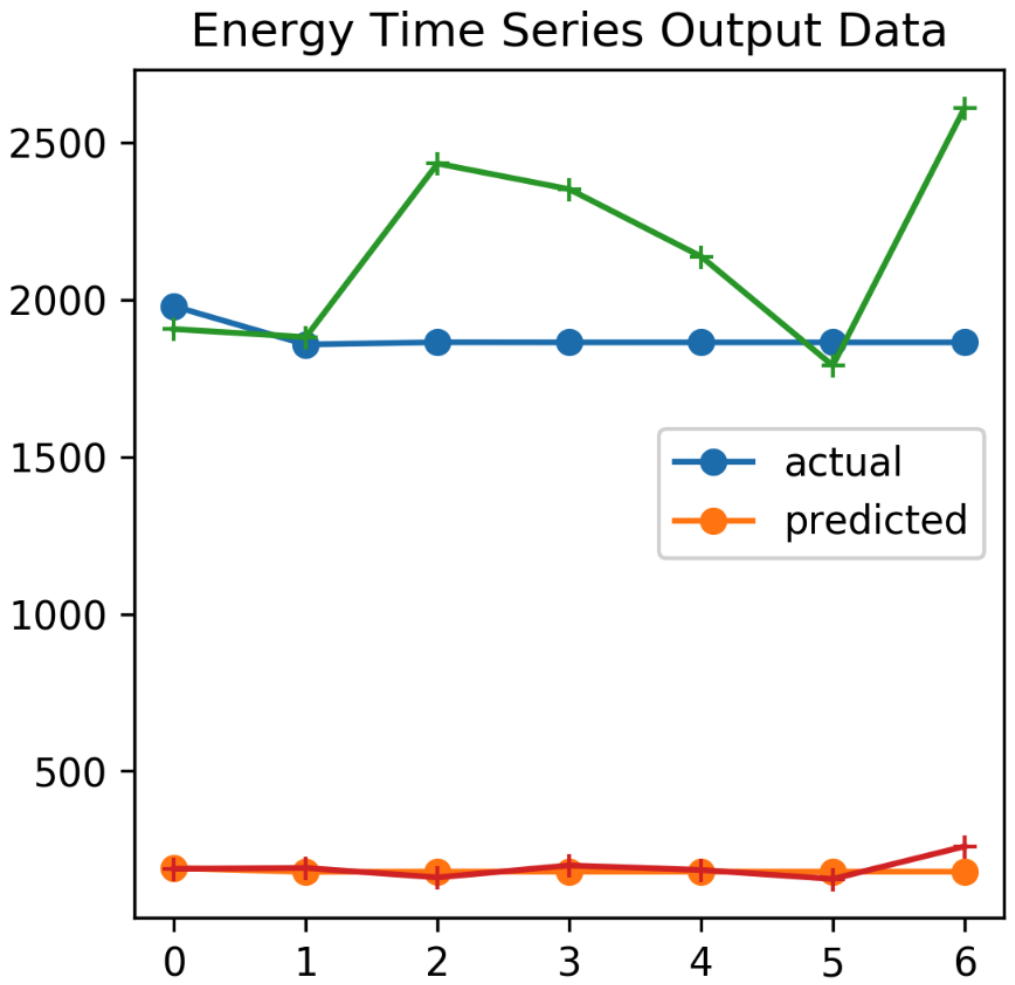
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 200)               167200
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 7, 200)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 7, 200)            320800
_________________________________________________________________
time_distributed_1 (TimeDist (None, 7, 100)            20100
_________________________________________________________________
time_distributed_2 (TimeDist (None, 7, 2)              202
=================================================================
Total params: 508,302
Trainable params: 508,302
Non-trainable params: 0
_________________________________________________________________
```

# Example of Input/Output Pair

```
In [84]: test_x, test_y = to_multivariate_supervised(test, n_input)

         print(np.shape(train_x))
         print(np.shape(train_y))
```

```
(1092, 14, 8)
(1092, 7, 2)
```



Thank you!

http://ibm.biz/model-exchange

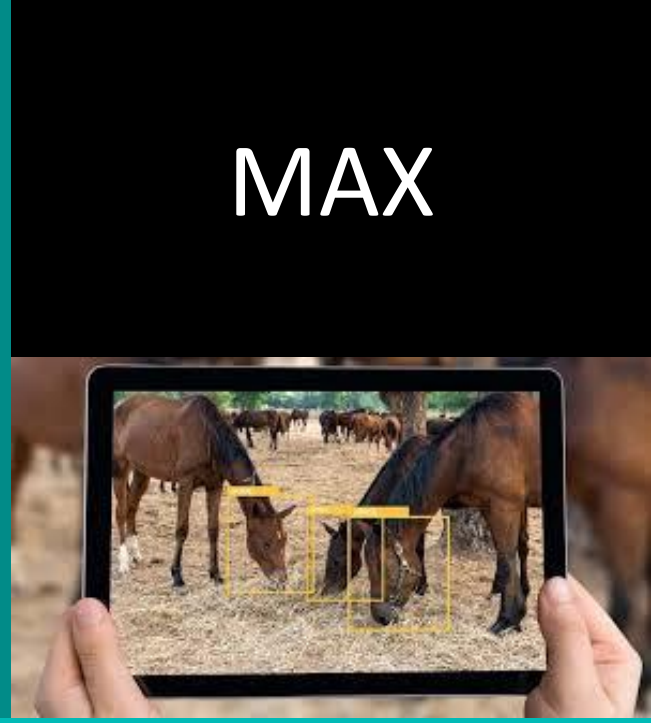
http://ibm.biz/max-developers

codait.org

twitter.com/JNilmeier

github.com/Nilmeier

developer.ibm.com


Sign up for IBM Cloud and try Watson Studio!

https://ibm.biz/BdY89J

https://datascience.ibm.com/