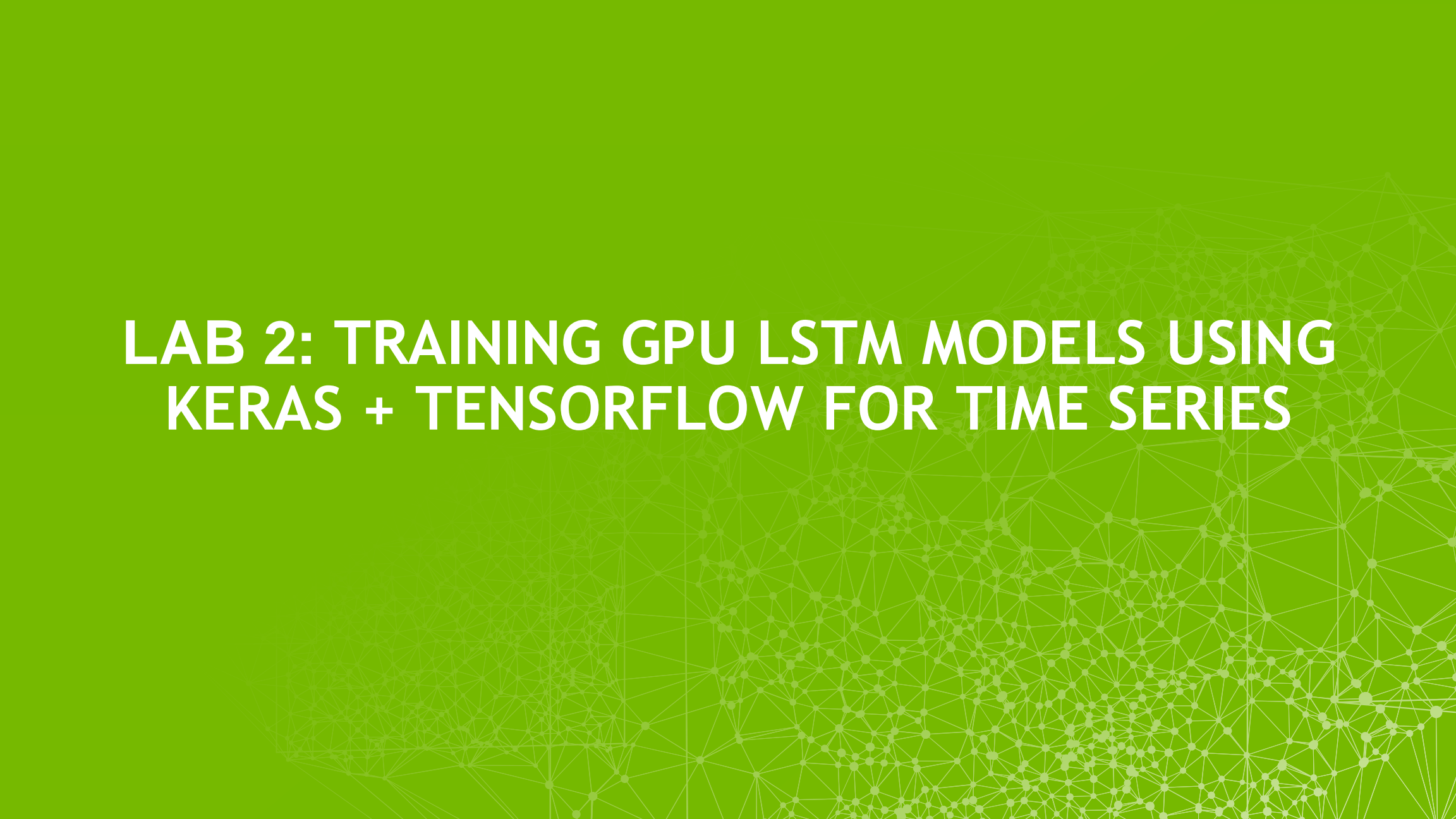


Applications of AI for Predictive Maintenance



DEEP
LEARNING
INSTITUTE

LAB 2: TRAINING GPU LSTM MODELS USING KERAS + TENSORFLOW FOR TIME SERIES



LAB 2 OVERVIEW

In this lab, we will leverage the Backblaze Hard Drive SMART data to train an LSTM model that will predict potential future failures. Unlike the previous lab, where the XGBoost model predicted based on current data, the LSTM model will leverage time series data to look for trends prior to a failure.

We will walk through creating the "normal" and "failing" sequences that will indicate when we expect to see early warning signs of a disk failure. Then we will use that data to train our model.

WORKSHOP STRUCTURE

LAB 1 (2 hrs.)

Training GPU XGBoost models with RAPIDS for Time Series

1. Discuss predictive maintenance, Blackblaze hard drive data, challenges of dealing with large noisy datasets.
2. Scope into short-term ML automation problem.
3. Ingest raw real dataset from GPU production line for XGBoost model.
4. Format DataFrames and into DMatrices to fit into model.
5. Analyze dataset statistics (i.e. false positives, true positives.)
6. Examine XGBoost performance over the model.
7. Learn how to balance imbalanced data and measure relevant KPIs to assess the model.

LAB 2 (2 hrs.)

Training GPU LSTM models using Keras+Tensorflow for Time Series

1. Discuss Recurrent Networks and Long Short-Term Memory (LSTMs).
2. Learn how to mitigate Vanishing Gradient Problem using LSTMs and get familiarized with their cell structure.
3. Learn how to create sequences of data to feed the temporal nature of RNNs.
4. Design different RNN architectures and even create your own model.
5. Learn how to create the confusion matrix and adjust the threshold to different f-1 scores.

LAB 3 (2 hrs.)

Training Autoencoder for Anomaly Detection

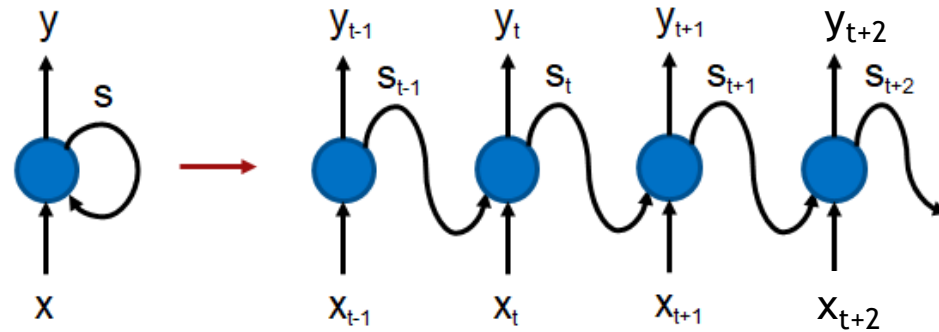
1. Focus on a different deep learning technique, which is called anomaly detection.
2. Learn how to use unsupervised learning algorithms like a deep autoencoder network to perform anomaly detection.
3. Learn how to create different autoencoder models in Keras.
4. General discussion on hyperparameter tuning and threshold settings.

RECURRENT NEURAL NETWORKS

The background of the slide features a smooth gradient transitioning from a deep green at the top to a bright yellow at the bottom. Overlaid on this gradient is a complex, abstract network of white lines and small circular nodes, resembling a neural network or a data visualization. The network is denser in the lower right portion of the image and fades out towards the top left.

Recurrent Neural Networks

- RNNs are created to solve the memory-persistence issue found in conventional feed forward networks.
- RNNs maintain a loop that allows the information to be persistent in the network.



- At each time step t , the network receives input X_t and outputs a value of y_t . The left side of the figure shows an RNN cell with input X and output y . There is also a loop-back of the state, which represents the memory. At a given timestep t , the output also depends on all previous timesteps $t-1, t-2, \dots, 0$.

VANISHING GRADIENT PROBLEM AND LSTMS

A simple RNN model only has a single hidden RNN layer while a stacked RNN model (needed for advanced applications) has multiple RNN hidden layers. A common problem in deep networks is the “vanishing gradient” problem, where the gradient gets smaller and smaller with each layer until it is too small to affect the deepest layers.

This is because small gradients or weights (values less than 1) are multiplied many times over through the multiple time steps, and the gradients shrink asymptotically to zero.

With the memory cell in LSTMs, we have continuous gradient flow (errors maintain their value) which thus eliminates the vanishing gradient problem and enables learning from sequences which are hundreds of time steps long.

LSTM MODEL

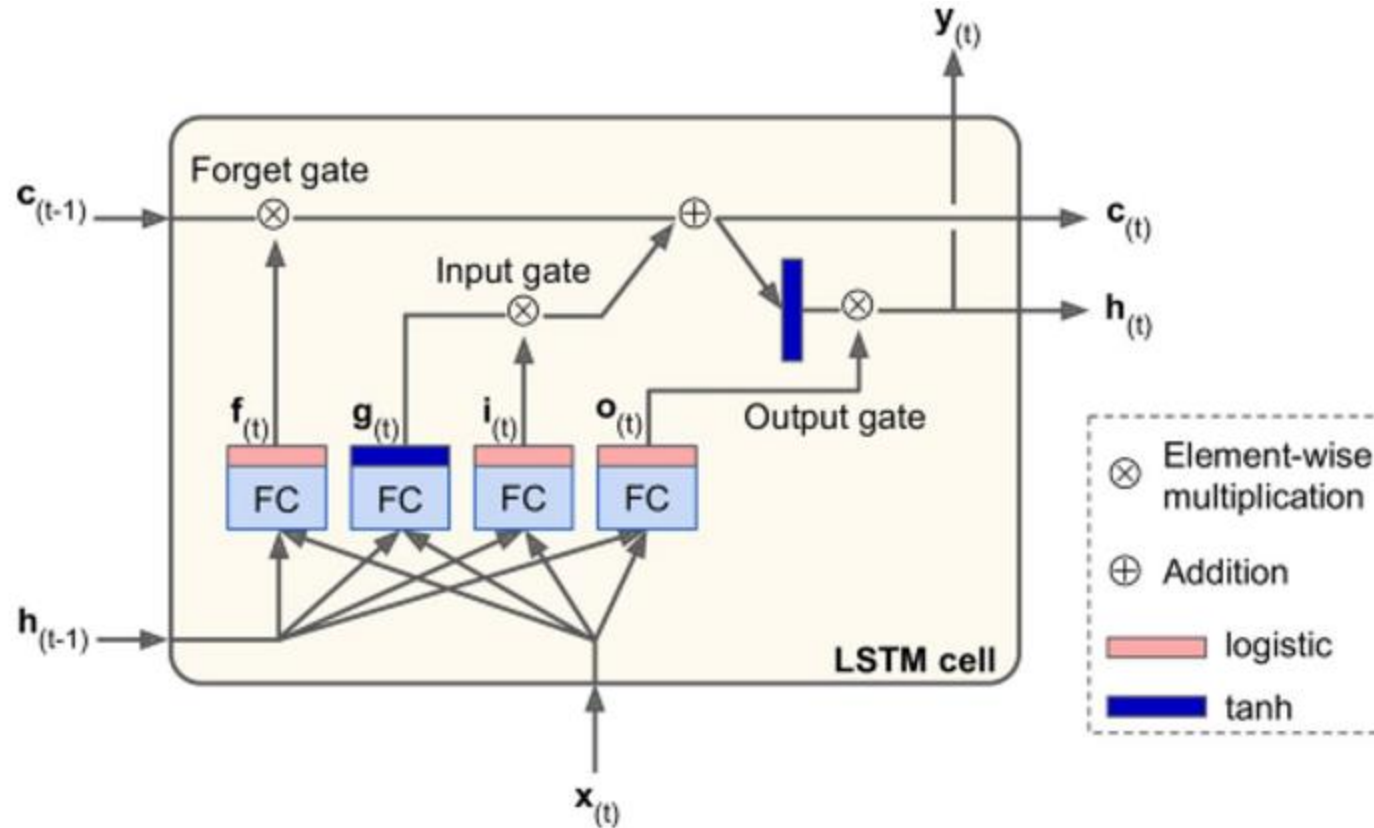


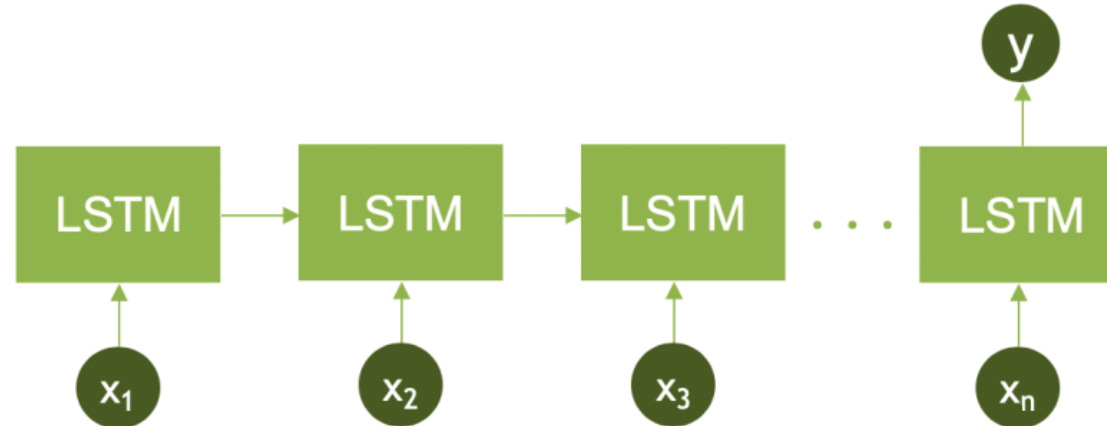
Image from book: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* by Aurélien Géron

LSTM APPLICATIONS

LSTMs, due to their ability to learn long term dependencies, are applicable to a number of sequence learning problems. These problems include language modeling and translation, acoustic modeling of speech, speech synthesis, speech recognition, audio and video data analysis, handwriting recognition and generation, sequence prediction, and protein secondary structure prediction.

In this lab, we apply the LSTM model to the Backblaze data we introduced in the previous lab to create a model that predicts the hard disk failure based on the recorded data-points at least a few days before the actual failure.

LSTM - CREATE SEQUENCES OF DATA BASED ON THE SERIAL NUMBER

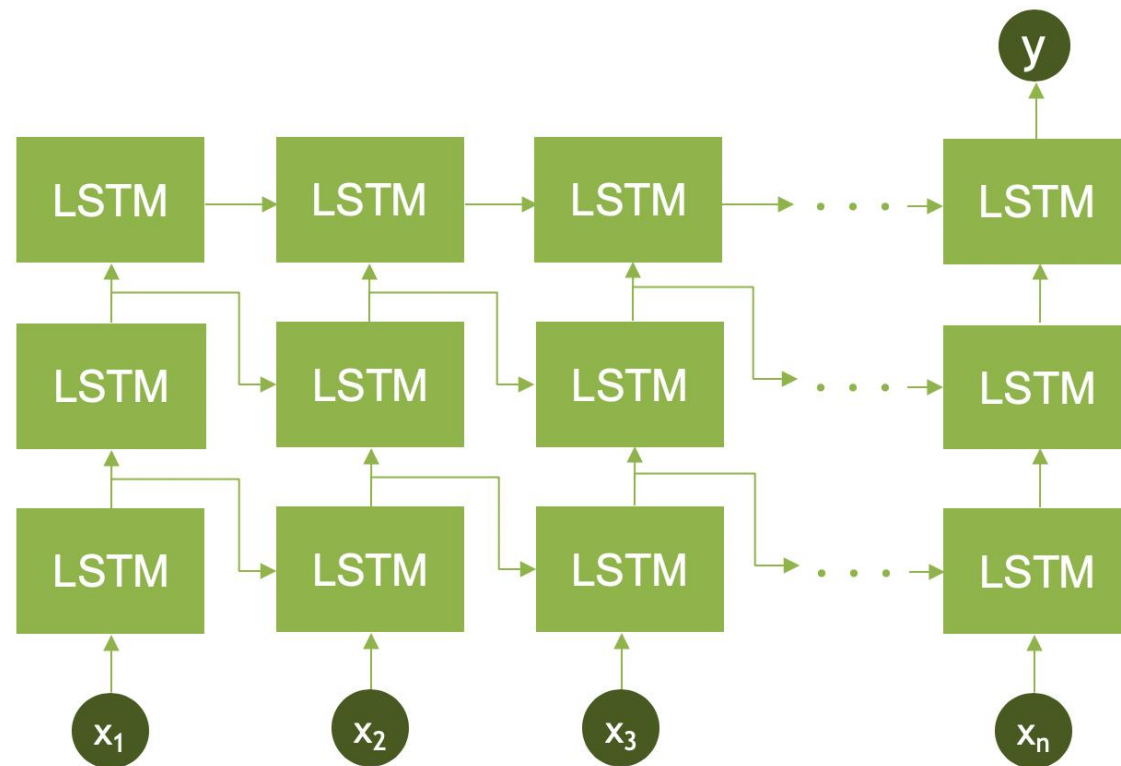


- Here we have n time steps representing each data point in the DataFrame.
- Also, only the last RNN (LSTM) cell is outputting the value y , which in our model indicates whether the data sequence ends up in a failure state or not. Note that the input values x_1, x_2, \dots, x_n are multi-dimensional values and not single scalars.

STACKED LSTMS

In reality, one-layer LSTM nodes are barely enough to capture the information encoded within a sequence. Alternatively, stacked LSTMs allow capturing more data complexity. In our model, we are going to use three layers of stacked LSTMs as shown in the next slide.

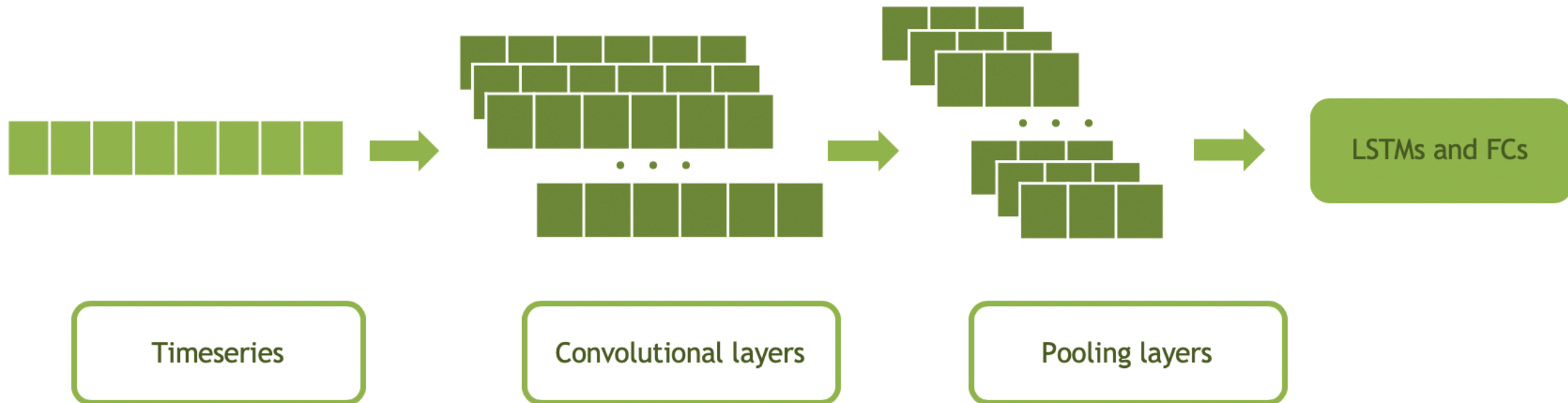
STACKED LSTMS



CNN LSTM ARCHITECTURE

The CNN LSTM uses a convolutional neural layer to extract features from the sequence on top of the LSTM layers, which also inherently extract features. Since we are dealing with a one dimensional time series, we are going to use 1D convolutional network, starting with a kernel size of 12 and 50 filters. We also use a max pooling layer of size 2 to shrink the dimension of the data.

CNN LSTM ARCHITECTURE





KERAS

KERAS

Keras: an API for specifying & training differentiable programs

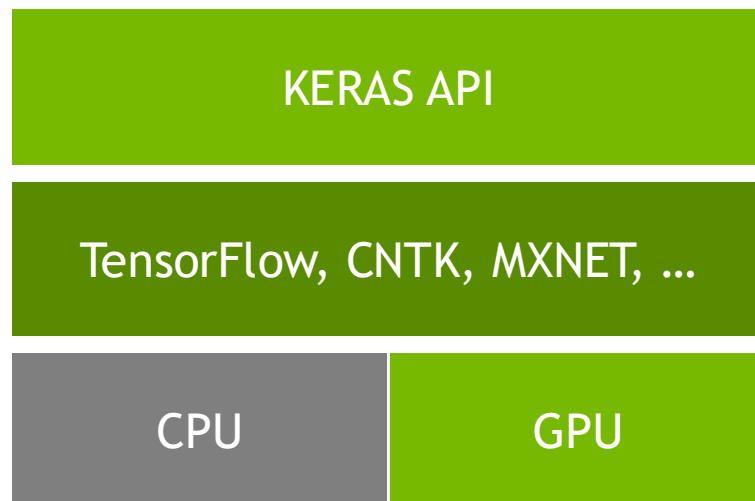
“Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*”

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.”

Source: <https://keras.io/>

KERAS

Keras: an API for specifying & training differentiable programs



KERAS - CODE SIMPLICITY

load model definition API

```
from keras.models import Sequential
```

```
model = Sequential()
```

load appropriate Keras layers and add to model

```
from keras.layers import Dense
```

```
model.add(Dense(units=64, activation='relu', input_dim=100))
```

```
model.add(Dense(units=10, activation='softmax'))
```

load appropriate optimizer and compile model

```
from keras.optimizers import SGD
```

```
optimizer1 = SGD(lr=0.01, momentum=0.9, nesterov=True)
```

```
model.compile(loss='categorical_crossentropy', optimizer=optimizer1, metrics=['accuracy'])
```

KERAS - CODE SIMPLICITY

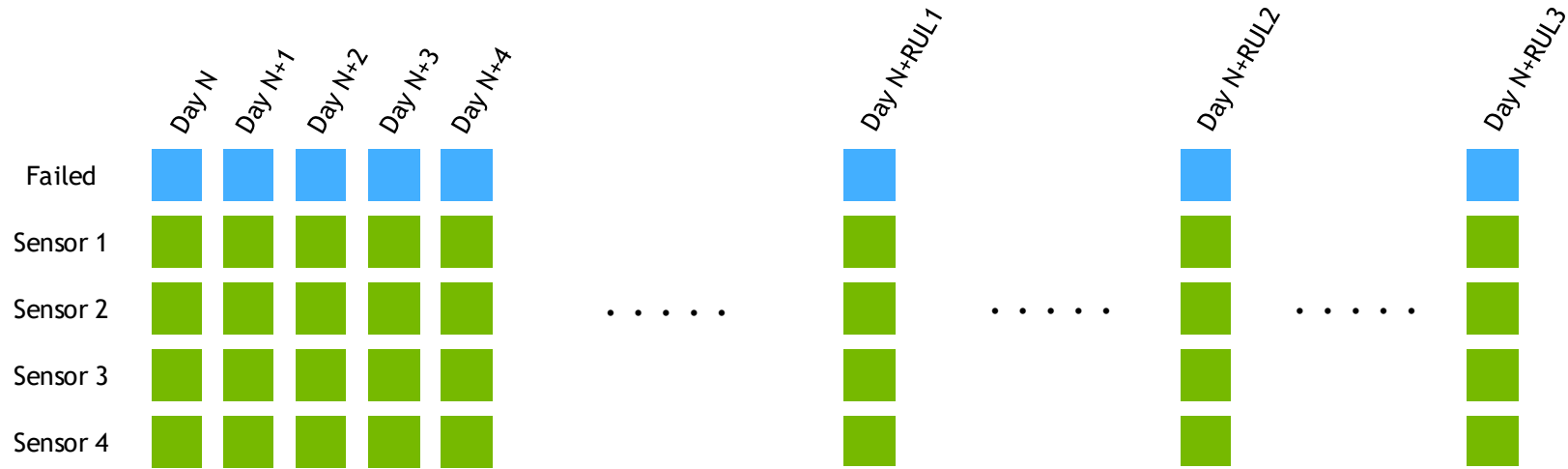
visualize model

model.summary()

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
dense_1 (Dense)	(None, 64)	6464
=====		
dense_2 (Dense)	(None, 10)	650
=====		
Total params: 7,114		
Trainable params: 7,114		
Non-trainable params: 0		
=====		

NOTE ABOUT LAB AND RUL



- For Lab 2, we will be training the model based with 5-day sequences and will use a 2-day look ahead to classify the sequence as failed or not.
- Typically, you would train multiple models using the failed state from a date in the future (Ex: 2 weeks out, 1 month out, 3 months out)

STARTING THE LAB

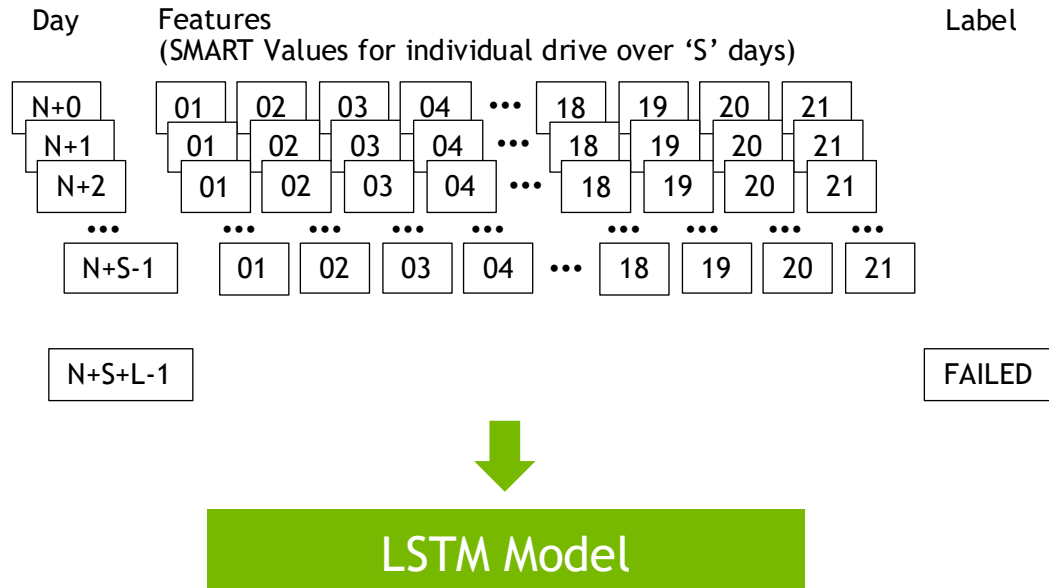
The background of the slide features a smooth gradient from a deep green on the left to a bright yellow on the right. Overlaid on this gradient is a complex, abstract network of white dots and thin white lines, resembling a molecular structure or a data network. The network is denser on the right side, where it also forms a faint, large-scale grid pattern.

LAB 2 MODEL SUMMARY - LSTM

Train LSTM Model with sequences of length 'S' to predict failure 'L' days in future

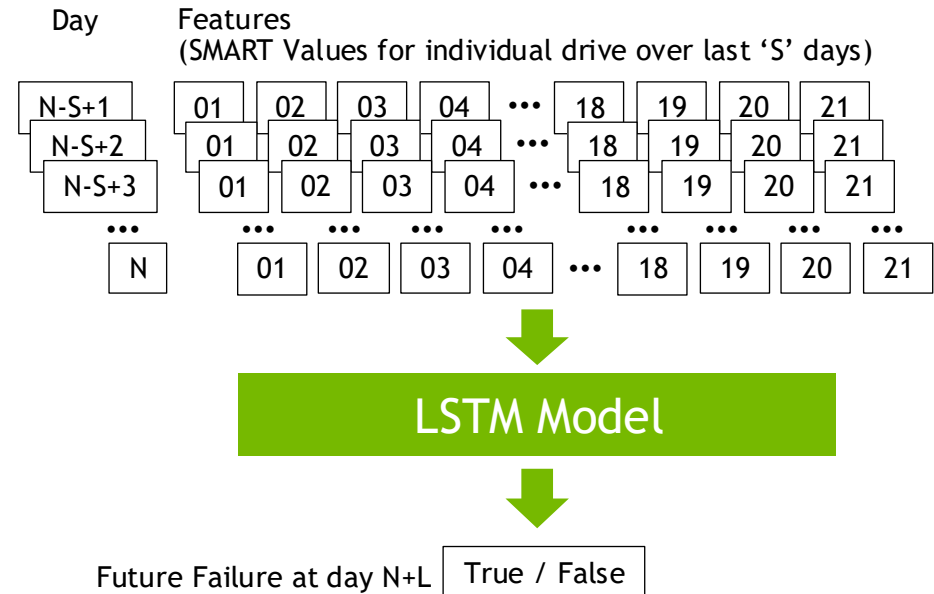
Training:

Use multi-day sequences of SMART features (from an individual hard drive) to train a LSTM model on whether a drive will fail 'L' days in the future



Inference:

Pass sequence of SMART values for an individual hard drive over the last 'S' days to see if drive will fail 'L' days in the future

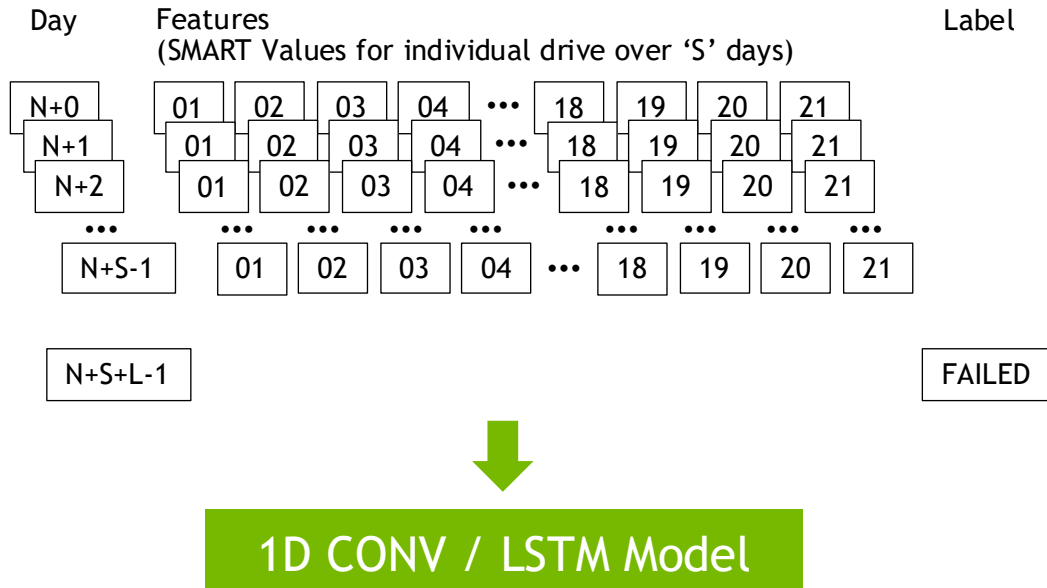


LAB 2 MODEL SUMMARY – 1D CONV / LSTM

Train LSTM Model with sequences of length 'S' to predict failure 'L' days in future

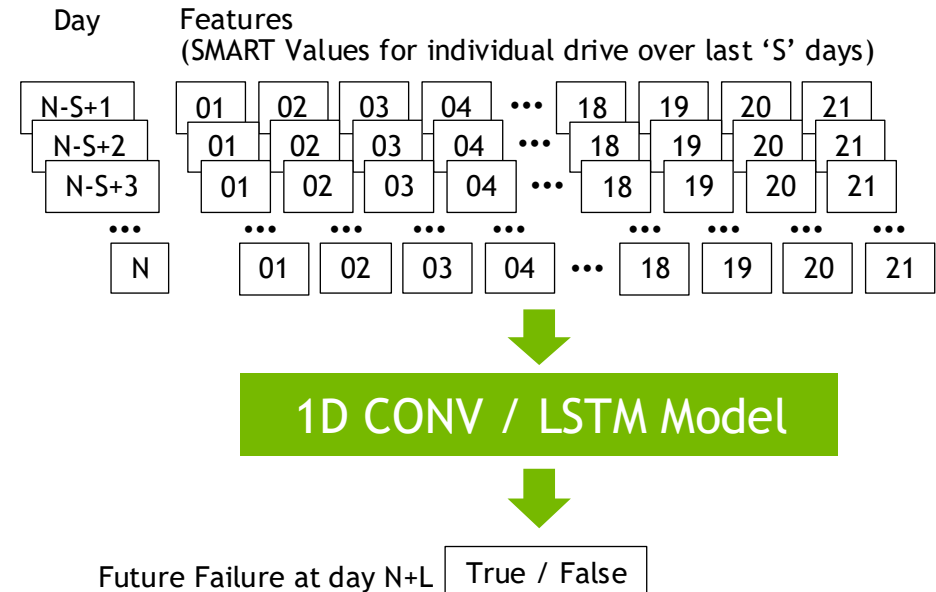
Training:

Use multi-day sequences of SMART features (from an individual hard drive) to train a 1D CONV / LSTM model on whether a drive will fail 'L' days in the future



Inference:

Pass sequence of SMART values for an individual hard drive over the last 'S' days to see if drive will fail 'L' days in the future



SUMMARY

The background of the slide features a smooth gradient from a vibrant green on the left to a clean white on the right. Overlaid on this gradient is a complex, abstract network of thin white lines connecting numerous small white dots, creating a mesh-like pattern that is denser on the right side.

LAB 2 SUMMARY

Training GPU LSTM models using Keras+Tensorflow for Time Series

1. Discuss Recurrent Networks and Long Short-Term Memory (LSTMs).
2. Learn how to mitigate Vanishing Gradient Problem using LSTMs and get familiarized with their cell structure.
3. Learn how to create sequences of data to feed the temporal nature of RNNs.
4. Design different RNN architectures and even create your own model.
5. Learn how to create the confusion matrix and adjust the threshold to different f-1 scores.

WHAT'S NEXT (LAB 3)?

Training Autoencoder for Anomaly Detection

1. Focus on a different deep learning technique, which is called anomaly detection.
2. Learn how to use unsupervised learning algorithms like a deep autoencoder network to perform anomaly detection.
3. Learn how to create different autoencoder models in Keras.
4. Discuss how to create and deploy GAN models for detecting anomalies.
5. General discussion on hyperparameter tuning and threshold settings.