AI in Finance
Big Data Analytics

Deep Learning for Financial Time Series Forecasting with TensorFlow

Min-Yuh Day, Ph.D.
Associate Professor
Department of Information Management
Tamkang University

http://mail.tku.edu.tw/myday

2019-11-26
## Course Schedule (1/2)

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Subject/Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2019/09/10</td>
<td>Course Orientation on AI in Finance Big Data Analytics</td>
</tr>
<tr>
<td>2</td>
<td>2019/09/17</td>
<td>AI in FinTech: Financial Services Innovation and Application</td>
</tr>
<tr>
<td>3</td>
<td>2019/09/24</td>
<td>ABC: AI, Big Data, Cloud Computing</td>
</tr>
<tr>
<td>4</td>
<td>2019/10/01</td>
<td>Business Models of Fintech</td>
</tr>
<tr>
<td>5</td>
<td>2019/10/08</td>
<td>Event Studies in Finance</td>
</tr>
<tr>
<td>6</td>
<td>2019/10/15</td>
<td>Case Study on AI in Finance Big Data Analytics I</td>
</tr>
<tr>
<td>7</td>
<td>2019/10/22</td>
<td>Foundations of AI in Finance Big Data Analytics with Python</td>
</tr>
<tr>
<td>8</td>
<td>2019/10/29</td>
<td>Case Study on Financial Industry Practice I</td>
</tr>
<tr>
<td>9</td>
<td>2019/11/05</td>
<td>Quantitative Investing with Pandas in Python</td>
</tr>
</tbody>
</table>
### Course Schedule (2/2)

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Subject/Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2019/11/12</td>
<td>Midterm Project Report</td>
</tr>
<tr>
<td>11</td>
<td>2019/11/19</td>
<td>Machine Learning in Finance Application with Scikit-Learn In Python</td>
</tr>
<tr>
<td>13</td>
<td>2019/12/03</td>
<td>Case Study on AI in Finance Big Data Analytics II</td>
</tr>
<tr>
<td>14</td>
<td>2019/12/10</td>
<td>Deep Learning for Financial Time Series Forecasting with TensorFlow II</td>
</tr>
<tr>
<td>15</td>
<td>2019/12/17</td>
<td>Case Study on Financial Industry Practice II</td>
</tr>
<tr>
<td>17</td>
<td>2019/12/31</td>
<td>Final Project Presentation I</td>
</tr>
<tr>
<td>18</td>
<td>2020/01/07</td>
<td>Final Project Presentation II</td>
</tr>
</tbody>
</table>
Deep Learning for Financial Time Series Forecasting with TensorFlow
Outline

• Deep Learning for Financial Time Series Forecasting with TensorFlow
  – Deep Learning
  – Financial Time Series Forecasting
  – TensorFlow
Aurélien Géron (2019),
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow:
Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition
O’Reilly Media, 2019

https://github.com/ageron/handson-ml2

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

Notebooks
1. The Machine Learning landscape
2. End-to-end Machine Learning project
3. Classification
4. Training Models
5. Support Vector Machines
6. Decision Trees
7. Ensemble Learning and Random Forests
8. Dimensionality Reduction
9. Unsupervised Learning Techniques
10. Artificial Neural Nets with Keras
11. Training Deep Neural Networks
12. Custom Models and Training with TensorFlow
13. Loading and Preprocessing Data
14. Deep Computer Vision Using Convolutional Neural Networks
15. Processing Sequences Using RNNs and CNNs
16. Natural Language Processing with RNNs and Attention
17. Representation Learning Using Autoencoders
18. Reinforcement Learning
19. Training and Deploying TensorFlow Models at Scale

https://github.com/ageron/handson-ml2
Sequences using RNNs and CNNs

```python
np.random.seed(43)
series = generate_time_series(1, 50 + 10)
X_new, Y_new = series[:, :50, :], series[:, 50:, :]
Y_pred = model.predict(X_new)[:, -1][..., np.newaxis]

plot_multiple_forecasts(X_new, Y_new, Y_pred)
plt.show()
```
An end-to-end open source machine learning platform

The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.

Get started with TensorFlow

https://www.tensorflow.org/
TensorFlow

• An end-to-end open source machine learning platform.
• The core open source library to help you develop and train ML models.
• Get started quickly by running Colab notebooks directly in your browser.

https://www.tensorflow.org/
Why TensorFlow 2.0

Why TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Easy model building
Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere
Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research
A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.
# TensorFlow 2.0 vs. 1.X

### TensorFlow 2.0

```python
outputs = f(input)
```

### TensorFlow 1.X

```python
outputs = session.run(f(placeholder), feed_dict={placeholder: input})
```

Source: [https://www.tensorflow.org/guide/effective_tf2](https://www.tensorflow.org/guide/effective_tf2)
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
                                      tf.keras.layers.Dense(128, activation='relu'),
                                      tf.keras.layers.Dropout(0.2),
                                      tf.keras.layers.Dense(10, activation='softmax')])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

https://www.tensorflow.org/overview/
TensorFlow 2 Quick Start

**TensorFlow 2 quickstart for beginners**

This short introduction uses Keras to:

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.

This is a Google Colaboratory notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select CONNECT.
2. Run all the notebook code cells: Select Runtime > Run all.

Download and install the TensorFlow 2 package. Import TensorFlow into your program:

```python
from __future__ import absolute_import, division, print_function, unicode_literals

# Install TensorFlow
try:
    # tensorflow_version only exists in Colab.
    tensorflow_version = 2.0
except Exception:
    pass
```

Basic classification: Classify images of clothing

This guide trains a neural network model to classify images of clothing, like sneakers and shirts. It's okay if you don't understand all the details; this is a fast-paced overview of a complete TensorFlow program with the details explained as you go.

This guide uses tf.keras, a high-level API to build and train models in TensorFlow.

```python
from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

https://www.tensorflow.org/tutorials/keras/classification
Image Classification
Fashion MNIST dataset

https://www.tensorflow.org/tutorials/keras/classification
Text classification with TensorFlow Hub: Movie reviews

This notebook classifies movie reviews as **positive** or **negative** using the text of the review. This is an example of **binary**—or two-class—classification, an important and widely applicable kind of machine learning problem.

The tutorial demonstrates the basic application of transfer learning with TensorFlow Hub and Keras.

We'll use the **IMDB dataset** that contains the text of 50,000 movie reviews from the **Internet Movie Database**. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are **balanced**, meaning they contain an equal number of positive and negative reviews.

This notebook uses **tf.keras**, a high-level API to build and train models in TensorFlow, and **TensorFlow Hub**, a library and platform for transfer learning. For a more advanced text classification tutorial using **tf.keras**, see the **MLCC Text Classification Guide**.

```python
from __future__ import absolute_import, division, print_function, unicode_literals
```
Text Classification with Pre Text

Text classification with preprocessed text: Movie reviews

This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.

We’ll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews.

This notebook uses tf.keras, a high-level API to build and train models in TensorFlow. For a more advanced text classification tutorial using tf.keras, see the MLCC Text Classification Guide.

Setup

```python
from __future__ import absolute_import, division, print_function, unicode_literals
```
Regression

Basic regression: Predict fuel efficiency

In a regression problem, we aim to predict the output of a continuous value, like a price or a probability. Contrast this with a classification problem, where we aim to select a class from a list of classes (for example, where a picture contains an apple or an orange, recognizing which fruit is in the picture).

This notebook uses the classic Auto MPG Dataset and builds a model to predict the fuel efficiency of late-1970s and early 1980s automobiles. To do this, we’ll provide the model with a description of many automobiles from that time period. This description includes attributes like: cylinders, displacement, horsepower, and weight.

This example uses the tf.keras API, see this guide for details.

```python
# Use seaborn for pairplot
!pip install -q seaborn

from __future__ import absolute_import, division, print_function, unicode_literals
import pathlib
```
TensorFlow 2.0
Time Series Forecasting

Time series forecasting

This tutorial is an introduction to time series forecasting using Recurrent Neural Networks (RNNs). This is covered in two parts: first, you will forecast a univariate time series, then you will forecast a multivariate time series.

```python
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```

https://www.tensorflow.org/tutorials/structured_data/time_series
Time Series Data

df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/
Time Series Data

\[ [100, 110, 120, 130, 140, 150] \]
Long Short Term Memory (LSTM) for Time Series Forecasting
## Time Series Data

\[ [10, 20, 30, 40, 50, 60, 70, 80, 90] \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10 20 30]</td>
<td>40</td>
</tr>
<tr>
<td>[20 30 40]</td>
<td>50</td>
</tr>
<tr>
<td>[30 40 50]</td>
<td>60</td>
</tr>
<tr>
<td>[40 50 60]</td>
<td>70</td>
</tr>
<tr>
<td>[50 60 70]</td>
<td>80</td>
</tr>
<tr>
<td>[60 70 80]</td>
<td>90</td>
</tr>
</tbody>
</table>
Deep Learning and Neural Networks
Deep Learning Foundations: Neural Networks
Deep Learning and Neural Networks

Input Layer (X)

Hidden Layer (H)

Output Layer (Y)
Deep Learning and Neural Networks

Input Layer (X)

Hidden Layers (H)

Output Layer (Y)

Deep Neural Networks
Deep Learning
Deep Learning
and
Deep Neural Networks
LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton.

"Deep learning."

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, conventional methods were effective mainly in low-dimensional data and are therefore applicable to many domains of science, business and government. In addition to beating records in image recognition\textsuperscript{1-4} and speech recognition\textsuperscript{5-7}, it has beaten other machine-learning techniques at predicting the activity of potential drug molecules\textsuperscript{8}, analysing particle accelerator data\textsuperscript{9,10}, reconstructing brain circuits\textsuperscript{11}, and predicting the effects of mutations in non-coding DNA on gene expression and disease\textsuperscript{12,13}. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding\textsuperscript{14}, particularly topic classification, sentiment analysis, question answering\textsuperscript{15} and language translation\textsuperscript{16,17}. 

Deep Learning

- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions
- Compatible with many variants of machine learning

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
What is Deep Learning?

- Loosely based on (what little) we know about the brain

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Neural Networks (NN)
A mostly complete chart of Neural Networks

Source: http://www.asimovinstitute.org/neural-network-zoo/
Deep Convolutional Network (DCN)

Deconvolutional Network (DN)

Deep Convolutional Inverse Graphics Network (DCIGN)

Generative Adversarial Network (GAN)

Liquid State Machine (LSM)

Extreme Learning Machine (ELM)

Echo State Network (ESN)

Deep Residual Network (DRN)

Kohonen Network (KN)

Support Vector Machine (SVM)

Neural Turing Machine (NTM)

Source: http://www.asimovinstitute.org/neural-network-zoo/
Convolutional Neural Networks
(CNN or Deep Convolutional Neural Networks, DCNN)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Recurrent Neural Networks (RNN)

Source: http://www.asimovinstitute.org/neural-network-zoo/
Long / Short Term Memory (LSTM)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Gated Recurrent Units (GRU)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Generative Adversarial Networks (GAN)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Support Vector Machines (SVM)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Neural networks (NN) 1960

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

X1

X2

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Multilayer Perceptrons (MLP) 1985

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
Support Vector Machine (SVM)
1995

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
Hinton presents the Deep Belief Network (DBN) New interests in deep learning and RBM State of the art MNIST 2005

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
Deep Recurrent Neural Network (RNN) 2009
Convolutional DBN
2010

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
Max-Pooling CDBN 2011

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
From image to text

A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

A group of *people* sitting on a boat in the water.

Convolutional Neural Networks (CNN)
Convolutional Neural Networks (CNN)

Architecture of LeNet-5 (7 Layers) (LeCun et al., 1998)


Convolutional Neural Networks (CNN)

- Convolution
- Pooling
- Fully Connection (FC) (Flattening)
A friendly introduction to Convolutional Neural Networks and Image Recognition

Convolution Layer

Pooling Layer

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-OI7ZB0MmU
A friendly introduction to Convolutional Neural Networks and Image Recognition

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-OJ7ZB0MmU
A friendly introduction to Convolutional Neural Networks and Image Recognition

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-Ol7ZB0MmU
A friendly introduction to
Convolutional Neural Networks and Image Recognition

Convolution Layer  Pooling Layer  Fully Connected Layer

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-OI7ZB0MmU
CNN Architecture

**CNN Convolution Layer**

Convolution is a mathematical operation to merge two sets of information

3x3 convolution

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Filter / Kernel

CNN Convolution Layer
Input x Filter --> Feature Map

receptive field: 3x3

<table>
<thead>
<tr>
<th>1x1</th>
<th>1x0</th>
<th>1x1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>1x1</td>
<td>1x0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x1</td>
<td>0x0</td>
<td>1x1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input x Filter

Feature Map

**CNN Convolution Layer**

**Input x Filter --> Feature Map**

receptive field: 3x3

<table>
<thead>
<tr>
<th>1</th>
<th>1x1</th>
<th>1x0</th>
<th>0x1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1x0</td>
<td>1x1</td>
<td>1x0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0x1</td>
<td>1x0</td>
<td>1x1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Input x Filter**

**Feature Map**

**CNN Convolution Layer**

Example convolution operation shown in 2D using a 3x3 filter.
CNN Convolution Layer

10 different filters  10 feature maps of size 32x32x1

final output of the convolution layer:

a volume of size 32x32x10

CNN Convolution Layer
Sliding operation at 4 locations

Source: Arden Dertat (2017), Applied Deep Learning - Part 4: Convolutional Neural Networks,
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
CNN Convolution Layer

two feature maps

CNN Convolution Layer

**Stride** specifies how much we move the convolution filter at each step.

Stride 1

Feature Map

**CNN Convolution Layer**

*Stride* specifies how much we move the convolution filter at each step.

**Stride 2**

*Feature Map*

---

CNN Convolution Layer

**Stride 1 with Padding**

- Stride 1 with Padding
- Feature Map

CNN Pooling Layer

Max Pooling

max pool with 2x2 window and stride 2

CNN Pooling Layer

CNN Architecture
4 convolution + pooling layers, followed by 2 fully connected layers

CNN Architecture

4 convolution + pooling layers, followed by 2 fully connected layers

https://gist.github.com/ardendertat/0fc5515057c47e7386fe04e9334504e3

```python
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1',
                 input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2), name='maxpool_1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
model.add(MaxPooling2D((2, 2), name='maxpool_2'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'))
model.add(MaxPooling2D((2, 2), name='maxpool_3'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'))
model.add(MaxPooling2D((2, 2), name='maxpool_4'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', name='dense_1'))
model.add(Dense(128, activation='relu', name='dense_2'))
model.add(Dense(1, activation='sigmoid', name='output'))
```
Dropout

Model Performance

Train Loss: 0.054, Val Loss: 1.345

Starts Overfitting

Train Accuracy: 0.981, Val Accuracy: 0.732

Visual Recognition

Image Classification
IS THIS A CAT or DOG?

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Convolutional Neural Networks
(CNNs / ConvNets)

http://cs231n.github.io/convolutional-networks/
A regular 3-layer Neural Network

input layer

hidden layer 1

hidden layer 2

output layer

http://cs231n.github.io/convolutional-networks/
A ConvNet arranges its neurons in three dimensions (width, height, depth)

http://cs231n.github.io/convolutional-networks/
The activations of an example ConvNet architecture.

http://cs231n.github.io/convolutional-networks/
ConvNets

32x32x3 CIFAR-10 image

first Convolutional layer

http://cs231n.github.io/convolutional-networks/
ConvNets

\[ x_0 \]

axon from a neuron

\[ w_0 \]

synapse

\[ w_0 x_0 \]

dendrite

\[ w_1 x_1 \]

\[ w_2 x_2 \]

\[ \sum_i w_i x_i + b \]

activation function

cell body

\[ f \left( \sum_i w_i x_i + b \right) \]

output axon

http://cs231n.github.io/convolutional-networks/
Convolution Demo

http://cs231n.github.io/convolutional-networks/
ConvNets

input volume of size [224x224x64]
is pooled with filter size 2, stride 2
into output volume of size [112x112x64]

http://cs231n.github.io/convolutional-networks/
ConvNets
max pooling

Single depth slice

max pool with 2x2 filters and stride 2

http://cs231n.github.io/convolutional-networks/
Convolutional Neural Networks (CNN) (LeNet)

Source: [http://deeplearning.net/tutorial/lenet.html](http://deeplearning.net/tutorial/lenet.html)
Recurrent Neural Networks (RNN)
Recurrent Neural Networks (RNN)

\[ y_t = f(x_t, y_{t-1}, h_{t-1}) \]
\[ h_t = g(x_t, y_{t-1}, h_{t-1}) \]

Input: \( x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2} \)
Output: \( y_{t-2}, y_{t-1}, y_t, y_{t+1}, y_{t+2} \)
Hidden: \( h_{t-2}, h_{t-1}, h_t, h_{t+1}, h_{t+2} \)
Recurrent Neural Networks (RNN)
Time Series Forecasting

Input

X_{t-2}  X_{t-1}  X_t  X_{t+1}  X_{t+2}

hidden

h_{t-2}  h_{t-1}  h_t  h_{t+1}  h_{t+2}

output

y_{t-2}  y_{t-1}  y_t  y_{t+1}  y_{t+2}
Recurrent Neural Networks (RNN)
Recurrent Neural Networks (RNN)
Sentiment Analysis

\[ X_t, X_{t-1}, X_{t-2}, X_{t+1}, X_{t+2} \]

\[ h_t, h_{t-1}, h_{t-2}, h_{t+1}, h_{t+2} \]

\[ y \]

This movie is very good
Recurrent Neural Networks (RNN) Sentiment Analysis

This movie is very boring
Recurrent Neural Network (RNN)

Course Description

Natural language processing (NLP) is one of the most important technologies of the information age. Understanding complex language utterances is also a crucial part of artificial intelligence. Applications of NLP are everywhere because people communicate most everything in language: web search, advertisement, emails, customer service, language translation, radiology reports, etc. There are a large variety of underlying tasks and machine learning models powering NLP applications. Recently, deep learning approaches have obtained very high performance across many different NLP tasks. These models can often be trained with a single end-to-end model and do not require traditional, task-specific feature engineering. In this spring quarter course students will learn to implement, train, debug, visualize and invent their own neural network models. The course provides a deep excursion into cutting-edge research in deep learning applied to NLP. The final project will involve training a complex recurrent neural network and applying it to a large scale NLP problem. On the model side we will cover word vector representations,
Recurrent Neural Networks (RNNs)
RNN

RNN long-term dependencies

I grew up in France... I speak fluent French.

Vanishing Gradient
Exploding Gradient
Recurrent Neural Networks (RNN)

\[ x_t - x_{t-1}, y_t - y_{t-1} \]

\[ h_t - h_{t-1}, y_{t+1} - y_t \]

\[ w \quad w \quad w \quad w \]

\[ v \quad v \quad v \quad v \]

Input \( x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2} \)

Output \( y_{t-2}, y_{t-1}, y_t, y_{t+1}, y_{t+2} \)

Hidden \( h_{t-2}, h_{t-1}, h_t, h_{t+1}, h_{t+2} \)
RNN

Vanishing Gradient problem
Exploding Gradient problem

if $|W| < 1$ (Vanishing)
if $|W| > 1$ (Exploding)
RNN

Vanishing Gradient problem

\[ W = 0.9 < 1 \text{ (Vanishing)} \]

RNN
Exploding Gradient problem

\[ \dot{X}_t = X_{t-1} - 1 \]
\[ \dot{Y}_t = Y_{t-1} - 2 \]
\[ \dot{H}_t = H_{t-1} - 2 \]

\[ W = 1.1 > 1 \text{ (Exploding)} \]

RNN LSTM

Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

Gated Recurrent Unit (GRU)
Gated Recurrent Unit (GRU)

reset gate
update gate

LSTM

**LSTM vs GRU**

**LSTM**

- $i$, $f$ and $o$ are the input, forget and output gates, respectively.
- $c$ and $c^\sim$ denote the memory cell and the new memory cell content.

**GRU**

- $r$ and $z$ are the reset and update gates, and $h$ and $h^\sim$ are the activation and the candidate activation.

Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

LSTM
Memory state (C)
**LSTM**

**forget gate** ($f$)

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$
**LSTM**

**input gate (i)**

\[
i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)
\]

\[
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\]

LSTM
Memory state (C)

\[ C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \]

LSTM
output gate (o)

\[ o_t = \sigma (W_o [h_{t-1}, x_t] + b_o) \]
\[ h_t = o_t \cdot \tanh (C_t) \]

**LSTM**

**forget (f), input (i), output (o) gates**

\[
\begin{align*}
    f_t &= \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\
    i_t &= \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\
    o_t &= \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)
\end{align*}
\]

Gated Recurrent Unit (GRU) update (z), reset (r) gates

\[
\begin{align*}
    z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
    r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
    \tilde{h}_t &= \tanh (W \cdot [r_t \cdot h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t
\end{align*}
\]

LSTM Recurrent Neural Network

- **Traditional Neural Network**
- **Music Generation**
- **Sentiment Classification**
- **Name Entity Recognition**
- **Machine Translation**

Source: https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent
Long Short Term Memory (LSTM) for Time Series Forecasting
The Sequence to Sequence model (seq2seq)

Source: http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/
Sequence to Sequence (Seq2Seq)

Source: https://google.github.io/seq2seq/
Transformer (Attention is All You Need) (Vaswani et al., 2017)

Transformer

INPUT

Je suis étudiant

THE TRANSFORMER

OUTPUT

I am a student

Transformer
Encoder Decoder

Transformer
Encoder Decoder Stack

Transformer
Encoder Self-Attention

Transformer Decoder

Transformer Encoder with Tensors

Word Embeddings

Transformer
Self-Attention Visualization

Transformer
Self-Attention Softmax Output

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT (Bidirectional Encoder Representations from Transformers)

Overall pre-training and fine-tuning procedures for BERT

Pre-training

Fine-Tuning

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT (Bidirectional Encoder Representations from Transformers)

**BERT input representation**

<table>
<thead>
<tr>
<th>Input</th>
<th>[CLS]</th>
<th>my</th>
<th>dog</th>
<th>is</th>
<th>cute</th>
<th>[SEP]</th>
<th>he</th>
<th>likes</th>
<th>play</th>
<th>##ing</th>
<th>[SEP]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Token Embeddings</th>
<th>$E_{[CLS]}$</th>
<th>$E_{my}$</th>
<th>$E_{dog}$</th>
<th>$E_{is}$</th>
<th>$E_{cute}$</th>
<th>$E_{[SEP]}$</th>
<th>$E_{he}$</th>
<th>$E_{likes}$</th>
<th>$E_{play}$</th>
<th>$E_{#ing}$</th>
<th>$E_{[SEP]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Embeddings</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
</tr>
<tr>
<td>Position Embeddings</td>
<td>$E_0$</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
<td>$E_7$</td>
<td>$E_8$</td>
<td>$E_9$</td>
<td>$E_{10}$</td>
</tr>
</tbody>
</table>

BERT, OpenAI GPT, ELMo

Fine-tuning BERT on Different Tasks

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

2 - **Supervised** training on a specific task with a labeled dataset.

**Semi-supervised Learning Step**

**Model:** BERT

**Dataset:** Predict the masked word (language modeling)

**Supervised Learning Step**

**Model:** BERT (pre-trained in step #1)

**Dataset:**

<table>
<thead>
<tr>
<th>Email message</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy these pills</td>
<td>Spam</td>
</tr>
<tr>
<td>Win cash prizes</td>
<td>Spam</td>
</tr>
<tr>
<td>Dear Mr. Atreides, please find attached...</td>
<td>Not Spam</td>
</tr>
</tbody>
</table>

Source: Jay Alammar (2019), The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), http://jalammar.github.io/illustrated-bert/
BERT Classification Input Output

BERT Classifier

Source: Jay Alammar (2019), The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), http://jalammar.github.io/illustrated-bert/
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

X1
X2

Source: https://www.youtube.com/watch?v=bxet2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
The Neuron

\[ x_1 \quad w_1 \rightarrow \quad x_2 \quad w_2 \rightarrow \quad \ldots \quad \ldots \rightarrow \quad x_n \quad w_n \rightarrow \quad y \]
Neuron and Synapse

Source: https://en.wikipedia.org/wiki/Neuron
The Neuron

\[ y = F\left(\sum_i w_i x_i\right) \]

\[ F(x) = \max(0, x) \]
\[ y = \max(0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3) \]
Neural Networks
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

X1 → H → Y
X2 → H → Y

Source: https://www.youtube.com/watch?v=bx2T-V8XR&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer (X)

Hidden Layers (H)

Output Layer (Y)

Deep Neural Networks
Deep Learning

Source: https://www.youtube.com/watch?v=bxe2T-V8XR&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer
(X)

Hidden Layer
(H)

Output Layer
(Y)

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer (X)  
Hidden Layer (H)  
Output Layer (Y)

Source: https://www.youtube.com/watch?v=bx2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Source: https://www.youtube.com/watch?v=P2HPcj8IRJE&list=PLjJh1vlSEYgvGod9wWiydumYl8hOXixNu&index=2
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
<table>
<thead>
<tr>
<th>Hours Sleep</th>
<th>Hours Study</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

Source: https://www.youtube.com/watch?v=bxet-V8XR&s=index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
<table>
<thead>
<tr>
<th>Hours Sleep</th>
<th>Hours Study</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Y = W X + b

Source: https://www.youtube.com/watch?v=G8eNWzOgqE
Output

\[ Y = WX + b \]

Weights

Trained

bias

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
\[ W X + b = Y \]

Scores → Probabilities

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
SoftMAX

$$W \times X + b = Y$$

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Logits → Scores → Probabilities

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{2.0}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.7$$

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{1.0}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.2$$

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{0.1}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.1$$

$$W X + b = Y$$

Logits $\rightarrow$ Scores $\rightarrow$ Probabilities

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
Training a Network

= Minimize the Cost Function

Source: https://www.youtube.com/watch?v=bxelV8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Training a Network

= Minimize the **Cost** Function

= Minimize the **Loss** Function

Source: https://www.youtube.com/watch?v=bxelV8XRsu&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y
Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bx2T-V8XR&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y

Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bx2T-V8XR9s&index=1&list=PLiaHh92iBX9hRr6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y
Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bxet2T-V8XR&s=index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Activation Functions
Activation Functions

**Sigmod**

**TanH**

**ReLU**

(Rectified Linear Unit)

\[ f(x) = \text{max}(0, x) \]

Activation Functions

**Sigmoid**

\[ f(x) = \frac{1}{1 + e^{-x}} \]

**TanH**

\[ \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \]

**ReLU**

\[ f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \]

Source: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/
Loss Function
Binary Classification: 2 Class

Activation Function: Sigmoid

Loss Function: Binary Cross-Entropy
Multiple Classification: 10 Class

Activation Function:
SoftMAX

Loss Function:
Categorical Cross-Entropy
Dropout

Dropout: a simple way to prevent neural networks from overfitting

(a) Standard Neural Net

(b) After applying dropout.

Learning Algorithm

While not done:

Pick a random training example "(input, label)"
Run neural network on "input"
Adjust weights on edges to make output closer to "label"

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
\[ y = \max (0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3) \]
Next time:

\[
y = \max(0, -0.23 \times x_1 + 0.31 \times x_2 + 0.65 \times x_3)
\]

\[
y = \max(0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3)
\]
Optimizer:
Stochastic Gradient Descent (SGD)

\[ J(w) \]

Initial weight

Global cost minimum

Gradient
This shows a function of 2 variables: real neural nets are functions of hundreds of millions of variables!
Gradient Descent
how neural networks learn

Average cost of all training data...

\[
\begin{align*}
(0.18 - 0.00)^2 + \\
(0.29 - 0.00)^2 + \\
(0.58 - 0.00)^2 + \\
(0.77 - 0.00)^2 + \\
(0.20 - 0.00)^2 + \\
(0.36 - 0.00)^2 + \\
(0.93 - 0.00)^2 + \\
(1.00 - 0.00)^2 + \\
(0.95 - 1.00)^2 + \\
(0.35 - 0.00)^2
\end{align*}
\]

What's the "cost" of this difference?

Source: 3Blue1Brown (2017), Gradient descent, how neural networks learn | Chapter 2, deep learning,
https://www.youtube.com/watch?v=IHZwWFHWa-w
Backpropagation

Source: 3Blue1Brown (2017), What is backpropagation really doing? | Chapter 3, deep learning, https://www.youtube.com/watch?v=Ilg3gGewQ5U
Learning Algorithm

While not done:

Pick a random training example “(input, label)”
Run neural network on “input”
Adjust weights on edges to make output closer to “label”

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Financial Time Series Forecasting
Time Series Data

AAPL

- Adj Close
- MA05
- MA20
- MA60
Time Series Data

\[ [100, 110, 120, 130, 140, 150] \]
Deep Learning with TensorFlow
Deep Learning Software

• TensorFlow
  – TensorFlow™ is an open source software library for high performance numerical computation.

• Keras
  – Deep Learning library for TensorFlow, CNTK

• PyTorch
  – An open source deep learning platform that provides a seamless path from research prototyping to production deployment.

• CNTK
  – Computational Network Toolkit by Microsoft Research

Source: http://deeplearning.net/software_links/
tf.keras

Keras:
High-level API for TensorFlow
Keras: The Python Deep Learning library

You have just found Keras.

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.

Use Keras if you need a deep learning library that:

- 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.

Read the documentation at Keras.io.

Keras is compatible with: Python 2.7-3.6.

http://keras.io/
F rom Research to Production

An open source deep learning platform that provides a seamless path from research prototyping to production deployment.

Get Started

http://pytorch.org/
Keras

• 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.

Source: https://keras.io/
TensorFlow

An end-to-end open source machine learning platform

Get started quickly by running Colab notebooks directly in your browser.

https://www.tensorflow.org/
TensorFlow

• An end-to-end open source machine learning platform.
• The core open source library to help you develop and train ML models.
• Get started quickly by running Colab notebooks directly in your browser.

https://www.tensorflow.org/
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10, activation='softmax')  
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

https://www.tensorflow.org/overview/
 TensorFlow 2 Quick Start

This short introduction uses Keras to:

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.

This is a Google Colaboratory notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select CONNECT.
2. Run all the notebook code cells: Select Runtime > Run all.

Download and install the TensorFlow 2 package. Import TensorFlow into your program:

```python
1 from __future__ import absolute_import, division, print_function, unicode_literals
2
3 # Install TensorFlow
4 try:
5  # @tensorflow_version only exists in Colab.
6  @tensorflow_version 2.x
7 except Exception:
8  pass
```

**TensorFlow 2**

**Time Series Forecasting**

This tutorial is an introduction to time series forecasting using Recurrent Neural Networks (RNNs). This is covered in two parts: first, you will forecast a univariate time series, then you will forecast a multivariate time series.

```python
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```

TensorFlow Playground

Tinker With a Neural Network Right Here in Your Browser. Don’t Worry, You Can’t Break It. We Promise.

http://playground.tensorflow.org/
TensorFlow is an Open Source Software Library for Machine Intelligence

https://www.tensorflow.org/
Tensor

- 3
  - # a rank 0 tensor; this is a scalar with shape []
- [1., 2., 3.]
  - # a rank 1 tensor; this is a vector with shape [3]
- [[1., 2., 3.], [4., 5., 6.]]
  - # a rank 2 tensor; a matrix with shape [2, 3]
- [[[1., 2., 3.]], [[7., 8., 9.]]]
  - # a rank 3 tensor with shape [2, 1, 3]

https://www.tensorflow.org/
Scalar 80

Vector $\begin{bmatrix} 50 & 60 & 70 \end{bmatrix}$

Matrix $\begin{bmatrix} 50 & 60 & 70 \\ 55 & 65 & 75 \end{bmatrix}$

Tensor $\begin{bmatrix} \begin{bmatrix} 50 & 60 & 70 \end{bmatrix} & \begin{bmatrix} 70 & 80 & 90 \end{bmatrix} \\ \begin{bmatrix} 55 & 65 & 75 \end{bmatrix} & \begin{bmatrix} 75 & 85 & 95 \end{bmatrix} \end{bmatrix}$
TensorFlow
TensorBoard

https://www.tensorflow.org/tensorboard/index.html#graphs
Deep Learning for Financial Time Series Forecasting
Deep Learning
for
Financial Market Prediction
Stock Market Prediction
Stock Price Prediction
Time Series Prediction
Time Series Data

```python
df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
```

<matplotlib.axes._subplots.AxesSubplot at 0x1150bac88>
Time Series Data

\[ [100, 110, 120, 130, 140, 150] \]
Long Short Term Memory (LSTM) for Time Series Forecasting
Time Series Data

\[ [10, 20, 30, 40, 50, 60, 70, 80, 90] \]

\[
\begin{array}{ccc}
[10 & 20 & 30] & 40 \\
[20 & 30 & 40] & 50 \\
[30 & 40 & 50] & 60 \\
[40 & 50 & 60] & 70 \\
[50 & 60 & 70] & 80 \\
[60 & 70 & 80] & 90 \\
\end{array}
\]
Deep Learning for Financial Time Series Forecasting

https://colab.research.google.com/drive/1aEK0eSev8Q-Y0nNY32geFk7CB8pVgSQM

```python
# univariate data preparation
from numpy import array
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Source: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
LSTM for Time Series Forecasting

```python
# univariate lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import matplotlib.pyplot as plt
#%matplotlib inline

# define dataset
X = array([[100, 110, 120], [110, 120, 130], [120, 130, 140], [130, 140, 150], [140, 150, 160]])
y = array([130, 140, 150, 160, 170])
# reshape from [samples, timesteps] into [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))

# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# fit model
history = model.fit(X, y, epochs=2000, verbose=0)
# demonstrate prediction
x_input = array([[150, 160, 170]])
x_input = x_input.reshape((1, 3, 1))
yhat = model.predict(x_input, verbose=0)
print('yhat:', yhat)
print(model.summary())
# list all data in history
print(history.history.keys())
# summarize history for loss
print('loss:', history.history['loss'][-1])
print('loss:', history.history['val_loss'][-1])
plt.plot(history.history['loss'])
plt.title('model loss')
yhat = model.predict(x_input, verbose=0)
print(yhat)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```
Deep Learning for Financial Time Series Forecasting

https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

```python
# univariate lstm examplerom numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import matplotlib.pyplot as plt
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
features = 1
X = X.reshape((X.shape[0], X.shape[1], features))
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
history = model.fit(X, y, epochs=500, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
print('yhat', yhat)
print(model.summary())
```
Deep Learning for Financial Time Series Forecasting

Using TensorFlow backend.

```
[[102.31296]]
yhat [[102.31296]]
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>lstm_1 (LSTM)</td>
<td>(None, 50)</td>
<td>10400</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 1)</td>
<td>51</td>
</tr>
</tbody>
</table>

Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0

```
None
dict_keys(['loss'])
loss: 0.000000
loss: 1.2578432517784677e-07
```
Deep Learning for Financial Time Series Forecasting

https://colab.research.google.com/drive/1aEK0eSev8Q-Y0nNY32geFk7CB8pVgSQM

Source: https://github.com/yash-1337/AAPL_LSTM_Stock_Predictor/blob/master/AAPL_daily_LSTM_stock_predictor.ipynb
Basic Classification

Fashion MNIST Image Classification

https://colab.research.google.com/drive/19PJOJi1vn1kjcutlzNHjRSLbeVl4kd5z

Train your first neural network: basic classification

This guide trains a neural network model to classify images of clothing, like sneakers and shirts. It's okay if you don't understand all the details; this is a fast-paced overview of a complete TensorFlow program with the details explained as we go.

This guide uses `tf.keras`, a high-level API to build and train models in TensorFlow.
Text Classification

IMDB Movie Reviews

https://colab.research.google.com/drive/1x16h1GhHsLrLYtPCvCHaoO1W-i_gror

Copyright 2018 The TensorFlow Authors.

Licensed under the Apache License, Version 2.0 (the "License");
MIT License

Text classification with movie reviews

Download the IMDB dataset
Explore the data
Convert the integers back to words
Prepare the data
Build the model
Hidden units
Loss function and optimizer
Create a validation set
Train the model
Evaluate the model

This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.

We'll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews.

This notebook uses tf.keras, a high-level API to build and train models in TensorFlow. For a more advanced text classification tutorial using tf.keras, see the MLCC Text Classification Guide.

```python
# memory footprint support libraries/code
!ln -s /opt/bin/nvidia-smi /usr/bin/nvidia-smi
!pip install gputil
!pip install psutil
!pip install humanize
import psutil
import humanize
import os
import GPUtil as GPU
GPUs = GPU.getGPUs()
gpu = GPUs[0]
def printmetrics():
   ...
```
Basic Regression
Predict House Prices

https://colab.research.google.com/drive/1v4c8ZHTnRtgdf2_25K_AURjR6SCVBRdlj

Predict house prices: regression

The Boston Housing Prices dataset

Examples and features

Labels

Normalize features

Create the model

Train the model

Predict

Conclusion

Copyright 2018 The TensorFlow Authors.

In a regression problem, we aim to predict the output of a continuous value, like a price or a probability. Contrast this with a classification problem, where we aim to predict a discrete label (for example, where a picture contains an apple or an orange).

This notebook builds a model to predict the median price of homes in a Boston suburb during the mid-1970s. To do this, we'll provide the model with some data points about the suburb, such as the crime rate and the local property tax rate.

This example uses the tf.keras API, see this guide for details.
# pip install pandas_datareader
2 import numpy as np
3 import pandas as pd
4 import pandas_datareader.data as web
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 import datetime as dt
8 %matplotlib inline
9
10 # Read Stock Data from Yahoo Finance
11 end = dt.datetime.now()
12 # start = dt.datetime(2018, 1, 1)
13 start = dt.datetime(2018, 1, 1)
14 df = web.DataReader('^TWII', 'yahoo', start, end)  # TWII #2330.TW #^DJI #AAPL
15 df.to_csv('TWII.csv')
16 print(df.head())
17 print(df.tail())
18 df2 = pd.read_csv('TWII.csv')  # df from_csv('AAPL.csv')
19 print(df2.tail())
20
21 df['Adj Close'].plot(legend=True, figsize=(12, 8), title='TWII', label='Adj Close')
22 plt.figure(figsize=(12, 9))
23 top = plt.subplot2grid((12, 9), (0, 0), rowspan=10, colspan=9)
24 bottom = plt.subplot2grid((12, 9), (10, 0), rowspan=2, colspan=9)
25 top.plot(df.index, df['Adj Close'], color='blue')  # df.index gives the dates
26 bottom.bar(df.index, df['Volume'])
27
28 # set the labels
29 top.axes.get_xaxis().set_visible(False)
30 top.set_title('TWII')
31 top.set_ylabel('Adj Close')
32 bottom.set_ylabel('Volume')
np.where
(df['MA20'] > df['MA60'], 12000, 9000)

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean()  # 5 days
df['MA20'] = df['Adj Close'].rolling(20).mean()  # 20 days
df['MA60'] = df['Adj Close'].rolling(60).mean()  # 60 days
df['Positions'] = np.where(df['MA20'] > df['MA60'], 12000, 9000)
df2 = pd.DataFrame({'Adj Close': df['Adj Close'], 'MA05': df['MA05'], 'MA20': df['MA20'], 'MA60': df['MA60'], 'Positions': df['Positions']})
df2.plot(figsize=(12, 9), legend=True, title='AAPL', secondary_y = 'Positions').legend(bbox_to_anchor=(1.2, 0.5))

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT
np.where(df['MA20'] > df['MA60'], 1, 0)

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean()  # 5 days
df['MA20'] = df['Adj Close'].rolling(20).mean()  # 20 days
df['MA60'] = df['Adj Close'].rolling(60).mean()  # 60 days
df['Positions'] = np.where(df['MA20'] > df['MA60'], 1, 0)
df2 = pd.DataFrame({'Adj Close': df['Adj Close'], 'MA05': df['MA05'], 'MA20': df['MA20'], 'MA60': df['MA60'], 'Positions': df['Positions']})
Yves Hilpisch (2018), Python for Finance: Mastering Data-Driven Finance, O'Reilly

https://github.com/yhilpisch/py4fi2nd

Source: https://www.amazon.com/Python-Finance-Mastering-Data-Driven/dp/1492024333
Aurélien Géron (2019),
O’Reilly Media, 2019

https://github.com/ageron/handson-ml2
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

https://github.com/ageron/handson-ml2
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

Notebooks
1. The Machine Learning landscape
2. End-to-end Machine Learning project
3. Classification
4. Training Models
5. Support Vector Machines
6. Decision Trees
7. Ensemble Learning and Random Forests
8. Dimensionality Reduction
9. Unsupervised Learning Techniques
10. Artificial Neural Nets with Keras
11. Training Deep Neural Networks
12. Custom Models and Training with TensorFlow
13. Loading and Preprocessing Data
14. Deep Computer Vision Using Convolutional Neural Networks
15. Processing Sequences Using RNNs and CNNs
16. Natural Language Processing with RNNs and Attention
17. Representation Learning Using Autoencoders
18. Reinforcement Learning
19. Training and Deploying TensorFlow Models at Scale

https://github.com/ageron/handson-ml2
Papers with Code
State-of-the-Art (SOTA)

Browse State-of-the-Art

- 1509 leaderboards • 1327 tasks • 1347 datasets • 17810 papers with code

Follow on Twitter for updates

Computer Vision

- Semantic Segmentation
  - 33 leaderboards
  - 667 papers with code

- Image Classification
  - 52 leaderboards
  - 564 papers with code

- Object Detection
  - 54 leaderboards
  - 467 papers with code

- Image Generation
  - 51 leaderboards
  - 231 papers with code

- Pose Estimation
  - 40 leaderboards
  - 231 papers with code

› See all 707 tasks

Natural Language Processing

- Machine Translation

- Language Modelling

- Question Answering

- Sentiment Analysis

- Text Generation

https://paperswithcode.com/sota
Papers with Code
Stock Market Prediction

Leaderboards
No evaluation results yet. Help compare methods by submit evaluation metrics.

Subtasks

Stock Price Prediction
3 papers with code

Stock Trend Prediction
2 papers with code

Stock Prediction
1 paper with code

https://paperswithcode.com/task/stock-market-prediction
The Quant Finance PyData Stack

- PyThalesians
- Zipline
- DX Analytics
- PyAlgoTrade
- QuantLib
- SciPy
- NumPy
- P[y]: IPython
- Python
- jupyter
- StatsModels: Statistics in Python
- matplotlib
- pandas
- scikits-image: Image processing in Python
- scikit-learn
- PyMC
- SymPy

Source: http://nbviewer.jupyter.org/format/slides/github/quantopian/pyfolio/blob/master/pyfolio/examples/overview_slides.ipynb#5
Summary

• Deep Learning for Financial Time Series Forecasting with TensorFlow
  – Deep Learning
  – Financial Time Series Forecasting
  – TensorFlow
• Ties de Kok (2017), Learn Python for Research, https://github.com/TiesdeKok/LearnPythonforResearch
• Data School (2015), Machine learning in Python with scikit-learn, https://www.youtube.com/playlist?list=PL5-da3qGB5iCeMbQuqbbCOQWcS6OYBr5A
• Deep Learning Basics: Neural Networks Demystified, https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
• Deep Learning SIMPLIFIED, https://www.youtube.com/playlist?list=PLjJh1vlSEYgYGod9wWiydumYl8hOXixNu
• 3Blue1Brown (2017), But what *is* a Neural Network? | Chapter 1, deep learning, https://www.youtube.com/watch?v=aircArvnKk
• 3Blue1Brown (2017), Gradient descent, how neural networks learn | Chapter 2, deep learning, https://www.youtube.com/watch?v=IHzwWFHwa-w
• 3Blue1Brown (2017), What is backpropagation really doing? | Chapter 3, deep learning, https://www.youtube.com/watch?v=Ilg3gGewQ5U
• Jay Alammar (2019), The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), http://jalammar.github.io/illustrated-bert/
• TensorFlow: https://www.tensorflow.org/
• Keras: http://keras.io/
• Udacity, Deep Learning, https://www.youtube.com/playlist?list=PLAwxTw4SYaPn_OWPFT9ulXLuQrlmzHfOV
• http://p.migdal.pl/2017/04/30/teaching-deep-learning.html
• https://github.com/leriomaggio/deep-learning-keras-tensorflow