AICE Global Perspectives AS - Science

Salvador Aleguas - Period 3

# 

**Title**

## A novel supervised regression approach using long short-term memory Algorithm for stock forecasting

# 

# 

# 

# 

# 

**Statement of purpose**

## The stock market is a volatile and extremely lucrative source of income. It powers a large part of the world’s economics, and has been known as being unpredictable and sheer luck. However, tools such as neural networks have given rise to algorithmic trading, which could potentially find a solution and pattern to a seemingly random money game. Neural networks are easy to create with limited resources and this project is to show the simplicity of creating a neural network to gain off the stock market.

# 

# 

**Introduction:**

Machine learning is often hailed as the most innovative discovery made in the 21st century and will be the technology of the future; however, machine learning has been prevalent for decades, even since the early 19th century. Ideas have been built upon each other, from abstract mathematical concepts to practical programs. This concept was first fully and experimentally tackled by Alan Turing, who came up with the famous question “Can computers think”. This spawned the ‘imitation game’ experiment that further developed on this topic.

Ideas developed over the years, with more mathematical foundations being built upon each other; however, it would still be a while from the 1950s when machine learning started to take the spotlight. Public awareness of AI spiked when an IBM computer named ‘deep blue’ beat a chess grandmaster.

Artificial intelligence is commonly associated to that of neural networks or deep learning; however, the terms are not synonymous. Artificial intelligence is an umbrella of concepts that covers the ability of machines to think and act rationally[[1]](#footnote-0), meaning that a chain of logic gates could be considered artificial intelligence. Deep learning and neural networks is a branch of artificial intelligence that is not solely algorithm based: it is code that writes and changes itself with given input data.

The three most widely used types of deep learning are: reinforcement, supervised, and unsupervised learning[[2]](#footnote-1). Reinforcement learning utilizes an agent to analyze and react to the environment it is in with the actions it can take, it is most commonly used in computerized simulations, as shown in examples where it can learn to play video games[[3]](#footnote-2). Supervised learning is the most common form of deep learning, where labeled input data is fed to a program to determine correlation and output. The data must be organized and split into both testing and training groups, a common ratio being 30% test and 70% training[[4]](#footnote-3). However, splitting up data can cause issues when data is limited. The more data you use for testing means the less data you have for training. This is used for regression and classification models, regression being the model used by this project. Unsupervised is training where a group of unlabeled data is clustered together and interpreted by data scientists.

Modern age computers are vastly superior to their outdated counterparts and therefore can handle a greater variety of tasks than our computers 30 years ago. As a result of this, deep learning, a much more efficient algorithm that uses much more data, is now a very practical application of processing power. Deep learning grows exponentially more in performance when fed data, and therefore can achieve high 99% accuracy with realistic amounts of data sets.

Stocks and stock predictions have been around almost as long as the concepts of machine learning; more people are familiarized with the concepts of stocks than machine learning, however. Stock prediction is so useful in financial returns that there are millions of people and jobs dedicated to predicting stock trends and how the market will change.

Machine learning is implemented using programming languages, a popular one being the interpreted language python, which will also be the language used in this project. Python innately is a slower language compared to its compiled counterparts, such as C; machine learning is a very CPU and GPU intensive process which means speed and efficiency is everything. Fortunately, python allows for a solution to this problem: C extensions.

There are many popular libraries that raise the abstraction of machine learning; the one that will be used in this project however is TensorFlow by Google. Tensorflow is written using C extensions to capitalize on speed, so one can use the speed of a hard-typed compiled language such as C with the abstraction of an interpreted language such as Python.

It is through a machine learning concept called regression that my program will attempt to predict the stock trends of each coming day; regression refers to the act of predicting data with a given input based on past data. Simplified, Deep learning works by creating a set amount of nodes and having parameters fed into each one of them; each node has a pseudo random value given and every node receives another, but equivalent , random value. The nodes then send their output data to other sets of nodes which repeat the process until the final set of nodes. The program learns by using past sets of data and training it against more data until it achieves high accuracy rates.

Of course, the more data available, the more accurate the network is. A reliable and consistent source will be required for this project to have live updates and feeds as well as historical data that goes back as far as possible. Yahoo Finance has historically been the choice for most programs until they deprecated their support for fetching data. Fortunately, some python libraries have solved this issue and implemented their own way to fetch data from Yahoo Finance, which is one of the required dependencies of this project.

Using machine learning and regression for stock prediction is not a new subject; in fact, a whole concept called algorithmic trading is based off of this concept and is quite prevalent today. The programs that exist are scarce and few, especially to that of public use. Stock prediction through neural networks is almost impossible to obtain without having to get past some sort of paywall. The data that is used to train as well as the program itself can have vastly different impacts on the outcome and prediction a neural network will make.

Self learning is important as well: a neural network should be able to correct, adjust, and fix itself based on errors in prediction and the like. This means that the best course of action would be to implement the neural network into a website with backend technology for not only ease of access but also functionality that comes from a 24/7 server that can constantly update from stock information.

**Hypothesis**:

## If a neural network is trained using the open, close, high, and low of a stock, then it will be able to predict the resulting values within 5% accuracy

# 

# 

**Materials**:

## Python interpreter version 3.6.8

## Computer with internet capabilities as well as ability to run python programs.

## Libraries used in appendix A

## Windows 10 operating system

## Python code used in appendix B

## Formad in appendix C

## Cost: $0

# 

**Procedures**:

1. Visting the link <https://github.com/saleguas/machine-learning-performance-analyzer/archive/master.zip> which will automatically download the required files to your computer
   1. The download will be in a compressed .zip file.
   2. Create a folder for the extracted files, name the folder ml-stock. The path to the folder will differ between computers, but what matters is that the folder and the contents are contact
      1. For example, my path is ‘C:/users/drale/documents/ml-stock
   3. The files must be uncompressed to be ran, to extract the files open up the downloaded .zip file by double clicking the file. After the file opens, press ctrl+A to highlight all files, then hold left mouse to drag them to any folder on the computer.
2. Visting the link <https://www.python.org/ftp/python/3.6.8/python-3.6.8-amd64.exe> will automatically download the python interpreter needed and listed in materials.
   1. Accept all defaults by clicking next on every window presented
3. Double click the python file called main.py to open up a GUI that is used to create the model
4. Enter 180 into the split input box and click start
   1. The program will run for a while and give a notification when it is completed.
5. Going into the /reports folder will have the reports with the day, month, year, and the time the report was created
6. Go into the desired report

**Observations**

The neural network being trained with minimal daily records of stock data could well prove the hypothesis with less than 1% error. The potency of such a small amount of data with the accuracy proves that algorithmic trading will if not already reinvent the way the stock market is being used and regulated, as normal people cannot compete against nearly perfect machines in predicting stock prices.

**Application**:

## Algorithmic trading can lead to heavy increases in financial status as well as showing the versatility of machine learning: being able to predict tomorrow’s stocks today. The neural network can adapt and change itself based on the new values every day.

**Recommendation**

It is hard to determine what exactly to do against algorithmic trading, as restricting it is an infringement of rights as well as intellectual freedom. However; the potency and effectiveness of machine learning is too strong and could potentially severely impact the economy. Companies should use current events or news to randomly fluctuate their prices in order to stop perfect predictions.

**Bibliography**

Domingos, Pedro. “A Few Useful Things to Know about Machine Learning.” *Pdfs.semanticscholar.org*, Communications of the ACM Volume 55 Issue 10, October 2012, https://pdfs.semanticscholar.org/c3b6/0802b56eeec611e9def0fdfbcaf42b851b99.pdf.

“The History of Machine Learning - BBC Academy.” *BBC News*, BBC, https://www.bbc.com/timelines/zypd97h.

“NVIDIA Blog: Supervised Vs. Unsupervised Learning.” *The Official NVIDIA Blog*, 20 Aug. 2019, https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/.

“What Is Artificial Intelligence? How Does AI Work?: Built In.” *What Is Artificial Intelligence? How Does AI Work? | Built In*, https://builtin.com/artificial-intelligence.

Lavanchy, Maude, and Amit Joshi. “An AI Taught Itself to Play a Video Game – for the First Time, It's Beating Humans.” *The Conversation*, 6 Aug. 2019, <http://theconversation.com/an-ai-taught-itself-to-play-a-video-game-for-the-first-time-its-beating-humans-118028>.

“The Next Generation of STEM Leaders.” *Society for Science the Public ISEF Pages*, [www.societyforscience.org/isef/](http://www.societyforscience.org/isef/).

**Appendix:**

1. Libraries used:

|  |
| --- |
| Keras==2.1.5 matplotlib==3.1.1 numpy==1.17.2 pandas==0.25.1 scikit-learn==0.21.3 tensorflow==1.5.0 |

1. Python code:

dataEditing.py

-------------

import pandas as pd

import numpy as np

# Reads in the csv file with the given path. If split is 0, then it's assumed not to be analyzing and instead predicting future values. Returns pandas dataframe with date and close value

def readData(path):

df = pd.read\_csv(path)

df['Date'] = pd.to\_datetime(df['Date'])

df = df.set\_index('Date')

print(df)

df = df.iloc[:, 3:4]

return df

# Splits the time series data at a given point and uses getPredictData on the left half and returns the right half as test data. Returns the left half, the x\_train, the y\_train, the x\_pred, the right half, and the dates

def getAnalyzeData(df, split):

train = df.iloc[:-split]

y\_true = df.iloc[-split:]

x\_train, y\_train, x\_pred, dates = getPredictData(train, split)

return train.values.reshape(-1, 1), x\_train.values.reshape(-1, 1), y\_train.values.reshape(-1, 1), x\_pred.values.reshape(-1, 1), y\_true.values.reshape(-1, 1), dates

# Shifts and edits the data and returns the x train, y train, the prediction input for the future values, and the dates of the future values as pandas series

def getPredictData(df, future):

train = df.iloc[:]

dates = pd.date\_range(df.index[-1], periods=future)

x\_pred = train[-future:]

x\_train = train[:-future]

y\_train = train[future:]

return x\_train, y\_train, x\_pred, dates

def preprocessData(x\_train\_raw, y\_train\_raw, x\_pred):

x\_train = []

y\_train = []

for i in range(30, len(x\_train\_raw)):

x\_train.append(x\_train\_raw[i-30, 0])

y\_train.append(y\_train\_raw[i-30, 0])

x\_train = np.array(x\_train).reshape(-1, 1)

y\_train = np.array(y\_train).reshape(-1, 1)

x\_pred = np.array(x\_pred).reshape(-1, 1)

print(x\_pred.shape)

x\_train = np.reshape(x\_train, (x\_train.shape[0], 1, x\_train.shape[1]))

y\_train = np.reshape(y\_train, (y\_train.shape[0], 1, y\_train.shape[1]))

x\_pred = np.reshape(x\_pred, (x\_pred.shape[0], 1, x\_pred.shape[1]))

return x\_train, y\_train, x\_pred

# arr = 1 2 3 4 5

# future = 1

# x\_pred = 5

# x\_train = 1 2 3 4

# y\_train = 2 3 4 5

externalData.py

-------------

from datetime import datetime

import os

import pandas as pd

import pandas\_datareader.data as web

import quandl

# Downloads the ticker given

def downloadData(ticker, path='../data/'):

symbol = 'WIKI/{}'.format(ticker)

df = web.DataReader(symbol, 'quandl', api\_key="xiypVZFNu6XEvRsCne29")

df.to\_csv(os.path.join(path, '{}.csv'.format(ticker)))

# Returns the path to the report folder. The type parameter is appended to the folder name

def reportPath(type):

dt\_string = datetime.now().strftime("%d-%m-%Y\_%H-%M-%S")

report\_base\_folder = os.path.join('..', 'reports')

report\_name = 'report\_{}\_{}'.format(type, dt\_string)

report\_path = os.path.join(report\_base\_folder, report\_name)

if not os.path.exists(report\_base\_folder):

os.makedirs(report\_base\_folder)

return report\_path

# Creates a report folder with the current data.

def createDataReport(allData, type):

report\_path = reportPath(type)

sheets\_path = os.path.join(report\_path, 'sheets')

rawSheets\_path = os.path.join(sheets\_path, 'raw')

analyzeSheets\_path = os.path.join(sheets\_path, 'anayze')

os.mkdir(report\_path)

os.mkdir(sheets\_path)

os.mkdir(rawSheets\_path)

for sheetName, sheet in allData.getRawSheets():

sheet.to\_csv(os.path.join(rawSheets\_path,

'{}\_raw\_data.csv'.format(sheetName)), index=False)

if allData.reportPossible():

os.mkdir(analyzeSheets\_path)

for sheetName, sheet in allData.getAnalyzeSheets():

sheet.to\_excel(os.path.join(

analyzeSheets\_path, '{}\_analyzed\_data.xlsx'.format(sheetName)), index=False)

if allData.forecastPossible():

forecastSheet = allData.generateForecastReport()

forecastSheet.to\_excel(os.path.join(

sheets\_path, 'data\_forecast\_analysis.xlsx'), index=False)

forecastSheet.to\_csv(os.path.join(

sheets\_path, 'data\_forecast\_analysis.csv'), index=False)

analyzeSheet = allData.generateAnalyzeReport()

analyzeSheet.to\_excel(os.path.join(

sheets\_path, 'all\_data\_analyzed.xlsx'), index=False)

analyzeSheet.to\_csv(os.path.join(

sheets\_path, 'all\_data\_analyzed.csv'), index=False)

Main.py

-------------

import argparse

import os

import modelCreation

import externalData

from gooey import Gooey

def getFilePaths(path):

paths = []

for root, dirs, files in os.walk(os.path.abspath(path)):

for file in files:

paths.append(os.path.join(root, file))

return paths

@Gooey

def main():

parser = argparse.ArgumentParser(

description='Interface for the stock prediction models. Every file in the data folder will be processed with the given command and report(s) are generated.')

parser.add\_argument(

'-d', '--data', help='Pass the Tickers of the stocks to download', metavar=("TICKER"), nargs='+')

parser.add\_argument('-i', '--input', help='Specify the input folder. By defult it is /data',

metavar=('LOCATION'), nargs=1)

# parser.add\_argument('-p', '--predict', help='Predict FUTURE values into the future for each stock',

# metavar=('FUTURE'), nargs=1)

parser.add\_argument('-a', '--analyze', help='Analyze the efficiency, splitting the time series data at SPLIT',

metavar=('SPLIT'), nargs=1)

args = vars(parser.parse\_args())

location = os.path.join('..', 'data')

if args['data']:

for arg in args['data']:

externalData.downloadData(arg)

if args['input']:

location = args['input'][0]

# if args['predict']:

# paths = getFilePaths(location)

# modelCreation.createPredictionProject(paths, int(args['predict'][0]))

if args['analyze']:

paths = getFilePaths(location)

modelCreation.createAnalyzeProject(paths, int(args['analyze'][0]))

main()

Modelcreation.py

---------------

#!/usr/bin/env python

# coding: utf-8

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

import os

import keras.backend as K

from keras.callbacks import EarlyStopping

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM

# Importing other files

import dataEditing

import externalData

from organizers import DatumHolder

# Creates a LTSM model with the gixen X and Y variables. Returns a keras model

def createModelPrediction(allData, split):

sc = MinMaxScaler(feature\_range=(0, 1))

allData\_scaled = sc.fit\_transform(allData)

training\_set\_scaled = allData\_scaled[:-split]

testing\_set\_scaled = allData\_scaled[-split:]

##########################################################

x\_train = []

y\_train = []

for i in range(60, len(training\_set\_scaled)):

x\_train.append(training\_set\_scaled[i - 60:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

x\_train, y\_train = np.array(x\_train), np.array(y\_train)

x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

##########################################################

inputs = allData\_scaled[len(allData\_scaled) -

len(testing\_set\_scaled) - 60:]

inputs = inputs.reshape(-1, 1)

print(inputs)

x\_pred = []

for i in range(60, len(inputs)):

x\_pred.append(inputs[i - 60:i, 0])

x\_pred = np.array(x\_pred)

x\_pred = np.reshape(x\_pred, (x\_pred.shape[0], x\_pred.shape[1], 1))

y\_true = sc.inverse\_transform(testing\_set\_scaled)

K.clear\_session()

early\_stop = EarlyStopping(monitor='loss', patience=1, verbose=1)

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True,

input\_shape=(x\_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50))

model.add(Dropout(0.2))

model.add(Dense(units=1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(x\_train, y\_train, epochs=100, batch\_size=32,

verbose=1, callbacks=[early\_stop])

y\_pred = model.predict(x\_pred)

y\_pred = sc.inverse\_transform(y\_pred)

return y\_pred, y\_true

# for s in range(1, 8):

# x\_pred['shift\_{}'.format(s)] = x\_pred.shift(s)

# x\_train['shift\_{}'.format(s)] = x\_pred.shift(s)

# y\_train['shift\_{}'.format(s)] = x\_pred.shift(s)

# # Creates a report with the future predictions

# def createPredictionProject(paths, future):

# allData = DatumHolder()

#

# for path in paths:

# fileName = os.path.basename(path)

# csvData = dataEditing.readData(path)

# x\_train, y\_train, x\_pred, dates = dataEditing.getPredictData(

# csvData, future)

#

# scaler = MinMaxScaler()

# scaler.fit(csvData.values.reshape(-1, 1))

# x\_train = scaler.transform(x\_train.values.reshape(-1, 1))

# y\_train = scaler.transform(y\_train.values.reshape(-1, 1))

# x\_pred = scaler.transform(x\_pred.values.reshape(-1, 1))

# # x\_train = pd.DataFrame(x\_train)

# # y\_train = pd.DataFrame(y\_train)

# # x\_pred = pd.DataFrame(x\_pred)

# #

# # for s in range(1, ROLLING\_WINDOW+1):

# # x\_pred['shift\_{}'.format(s)] = x\_pred.iloc[:, 0].shift(s)

# # x\_train['shift\_{}'.format(s)] = x\_train.iloc[:, 0].shift(s)

# #

# # x\_pred = x\_pred.dropna().values

# # x\_train = x\_train.dropna().values

# # y\_train = y\_train.dropna().iloc[:-ROLLING\_WINDOW, ].values

# x\_pred = x\_pred[:, None]

# y\_pred = createModelPrediction(x\_train, y\_train, x\_pred)

# y\_pred = scaler.inverse\_transform(y\_pred)

#

# y\_pred = y\_pred.flatten()

#

# allData.addRawSheet(fileName, y\_pred, dates)

#

# externalData.createDataReport(allData, 'predict')

# Creates a report analyzing the model with a given split

def createAnalyzeProject(paths, split):

dataHolder = DatumHolder()

for path in paths:

fileName = os.path.basename(path)

csvData = dataEditing.readData(path)

dates = csvData.index[-split:]

allData = csvData.values

y\_pred, y\_true = createModelPrediction(allData, split)

dataHolder.addAnalyzeSheet(fileName, y\_pred.flatten(), y\_true.flatten(), dates)

externalData.createDataReport(dataHolder, 'analyze')

Organizers.py

------------

import numpy as np

import pandas as pd

class Datum:

def \_\_init\_\_(self, name, data):

self.name = name

self.data = data

class DatumHolder:

def \_\_init\_\_(self):

self.rawSheets = []

self.analyzeSheets = []

def addRawSheet(self, name, y\_pred, dates):

sheet = pd.DataFrame()

sheet['Predicted Close'] = y\_pred

sheet['Date'] = dates

datumSheet = Datum(name, sheet)

self.rawSheets.append(datumSheet)

def addAnalyzeSheet(self, name, y\_pred, y\_true, dates):

print(y\_pred.shape)

print(y\_true.shape)

sheet = pd.DataFrame()

sheet['Date'] = dates

sheet['Days'] = np.arange(1, len(dates)+1)

sheet['Predicted Close'] = y\_pred

sheet['Actual Close'] = y\_true

sheet['relative % error'] = (sheet['Predicted Close'] - sheet['Actual Close']) / sheet['Predicted Close'] \* 100

sheet['absolute % error'] = abs(y\_true - y\_pred) / y\_pred \* 100

print(sheet.head())

datumSheet = Datum(name, sheet)

self.analyzeSheets.append(datumSheet)

def getRawSheets(self):

for datum in self.rawSheets:

yield datum.name, datum.data

def getAnalyzeSheets(self):

for datum in self.analyzeSheets:

yield datum.name, datum.data

def generateAnalyzeReport(self):

sheet = pd.DataFrame()

names = [datum.name for datum in self.analyzeSheets]

datesFrom = [datum.data['Date'].iloc[0]

for datum in self.analyzeSheets]

datesTo = [datum.data['Date'].iloc[-1] for datum in self.analyzeSheets]

averagePredictedClose = [

datum.data['Predicted Close'].mean() for datum in self.analyzeSheets]

averageActualClose = [datum.data['Actual Close'].mean()

for datum in self.analyzeSheets]

averageRelError = [datum.data['relative % error'].mean()

for datum in self.analyzeSheets]

averageAbsError = [datum.data['absolute % error'].mean()

for datum in self.analyzeSheets]

sheet['Name'] = names

sheet['Starting Date'] = datesFrom

sheet['Ending Date'] = datesTo

sheet['Average Predicted Close'] = averagePredictedClose

sheet['Average Actual Close'] = averageActualClose

sheet['Average Percent Relative Error'] = averageRelError

sheet['Average Percent Absolute Error'] = averageAbsError

return sheet

def generateForecastReport(self):

sheet = pd.DataFrame()

names = [datum.name for datum in self.analyzeSheets]

oneDayRelError = [datum.data['relative % error'].iloc[0].mean() for datum in self.analyzeSheets]

oneWeekRelError = [datum.data['relative % error'].iloc[:7].mean() for datum in self.analyzeSheets]

oneMonthRelError = [datum.data['relative % error'].iloc[:30].mean() for datum in self.analyzeSheets]

threeMonthRelError = [datum.data['relative % error'].iloc[:90].mean() for datum in self.analyzeSheets]

sixMonthRelError = [datum.data['relative % error'].iloc[:180].mean() for datum in self.analyzeSheets]

oneDayAbsError = [datum.data['absolute % error'].iloc[0].mean() for datum in self.analyzeSheets]

oneWeekAbsError = [datum.data['absolute % error'].iloc[:7].mean() for datum in self.analyzeSheets]

oneMonthAbsError = [datum.data['absolute % error'].iloc[:30].mean() for datum in self.analyzeSheets]

threeMonthAbsError = [datum.data['absolute % error'].iloc[:90].mean() for datum in self.analyzeSheets]

sixMonthAbsError = [datum.data['absolute % error'].iloc[:180].mean() for datum in self.analyzeSheets]

sheet['Name'] = names

sheet['1 day Rel'] = oneDayRelError

sheet['1 day Abs'] = oneDayAbsError

sheet['1 week Rel'] = oneWeekRelError

sheet['1 week Abs'] = oneWeekAbsError

sheet['1 month Rel'] = oneMonthRelError

sheet['1 month Abs'] = oneMonthAbsError

sheet['3 month Rel'] = threeMonthRelError

sheet['3 month Abs'] = threeMonthAbsError

sheet['6 month Rel'] = sixMonthRelError

sheet['6 month Abs'] = sixMonthAbsError

return sheet

def reportPossible(self):

return len(self.analyzeSheets) != 0

def forecastPossible(self):

return min([len(datum.data['relative % error']) for datum in self.analyzeSheets])

1. Required format:
   1. The data must be in a csv file
   2. The data must have the following fields
      1. Date
      2. Closing

**Abstract**:

Machine Learning has been around for years, and still computer scientists and big companies haven't found a sure way to gain profit in the stock market. Our stock prediction algorithm uses a library called Keras, and has been relatively successful. The main stocks that we tested were AMD, AMT, AMZN, ARNC, DISCA, FB, FOXA, RMD, and WMT. These stocks have been chosen intentionally, as they serve as good markers for increasing, volatile, and stagnant data. The purpose of this experiment is to demonstrate how the seemingly random and volatile stock market can be predicted using deep learning with minimal training data. Our model achieved an average absolute percent error of 6.18% throughout every stock tested, proving a high accuracy for even unpredictable stocks.

1. “What Is Artificial Intelligence? How Does AI Work?: Built In.” *What Is Artificial Intelligence? How Does AI Work? | Built In*, https://builtin.com/artificial-intelligence. [↑](#footnote-ref-0)
2. “NVIDIA Blog: Supervised Vs. Unsupervised Learning.” *The Official NVIDIA Blog*, 20 Aug. 2019, https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/. [↑](#footnote-ref-1)
3. Lavanchy, Maude, and Amit Joshi. “An AI Taught Itself to Play a Video Game – for the First Time, It's Beating Humans.” *The Conversation*, 6 Aug. 2019, http://theconversation.com/an-ai-taught-itself-to-play-a-video-game-for-the-first-time-its-beating-humans-118028. [↑](#footnote-ref-2)
4. [↑](#footnote-ref-3)