INSTIUTO TECNOLOGICO DE ESTUDIOS SUPERIORES DE MONTERREY

# Intelligent Systems:

"Number recognition with neural network"

Jose Carlos Pacheco Sánchez – A01702828

*ITESM- Qro - 2021*

# CONTENT TABLE:

Cover page……...1

Content table……2

General concepts…….…….3

Starting ……4

Understanding the path……5

Neuron Layers……..6

Training Model…….7

TensorFlow + Keras ……12

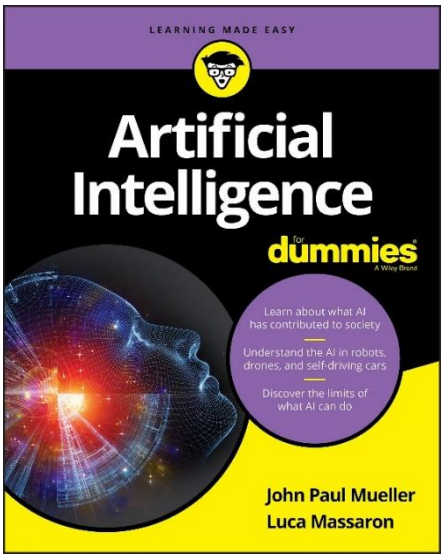
Code ……14

Results……..19

References……20

GENERAL CONCEPTS:

First, we need to identify what is Image recognition and for what is used for:




*"Optical Character Recognition is a process of digitizing texts from images of symbols or characters that belong to a certain alphabet."* This means that thanks to this, the data can be identified and stored from the images and thus be able to interact with these characters.

Knowing this, we can think in many and different applications on this specific field, but What do we use to do something like that?

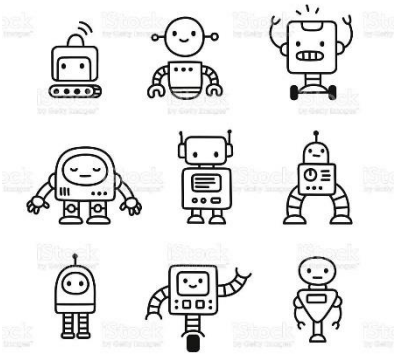
The answer is kind strange but simple: *Artificial neural networks* or more know as neural networks, they are based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons.

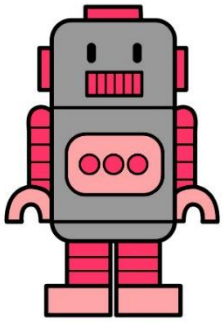
There are many applications for this filed, from easy recognition, text or cancer detection, In this project we will go through an especially useful and relatively simple point: image recognition, more specifically number recognition


To do this we will use two specific tools, Python and keras TensorFlow framework

STARTING:

To achieve all this, we need to have a little notion of how neural networks work and more specifically what is the general process of learning. Roughly the creation of a neural network is like the construction of a robot, the first thing we will use will be the pieces to be able to assemble it, for its just having the notion that a robot has a body, arms, and a head; of course, there will always be variants and different ways of putting them in a specific order.





In this case we will be using a template that we know that it works. The process will look like this.

1.- Receive a handwritten number as input

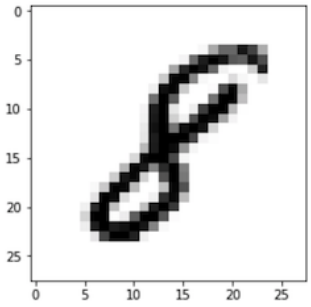
2.- Process the neuronal network

3.- As output it will give us the number that represents

UNDERSTADNIN THE PATH:

But how can we do this if its not a real person? It’s a program.

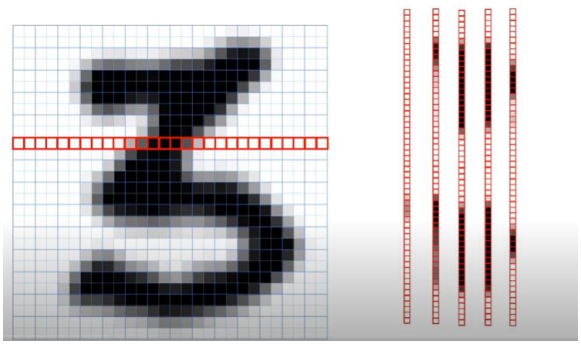
We are going to take the images and separate it in pixels, and assign the values into black and white, where 0 is black and 255 is white.




For this project we are going to use the MNSIT Dataset Modified *(National Institute of Standards and Technology),* MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. Its usually used for:

- Computer vision fundamentals including simple neural networks.
- Classification methods such as SVM and K-nearest neighbors

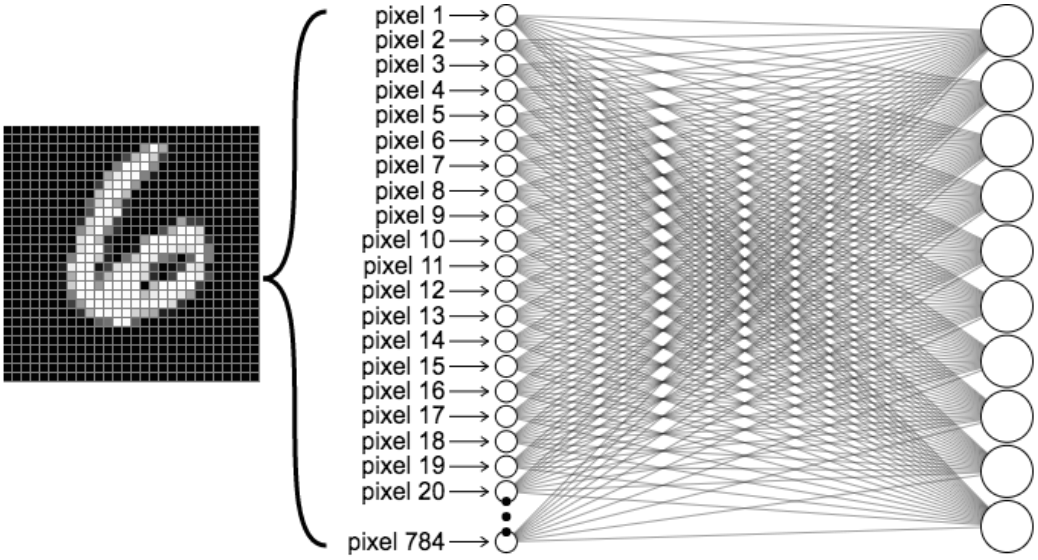
For this case we are going to use one of the algorithms to implement the neural network and using the images of MNSIT that are of 28 x28 pixels (784 pixels) we got something like this.



NEURAL LAYERS:

Now we need to understand how we are going to design our layers and number o neurons, but this is kind of “simple.”

We know that there are 784 pixels, that means we are going to use a least 784 neurons as our input payer to analyze each of the pixel.



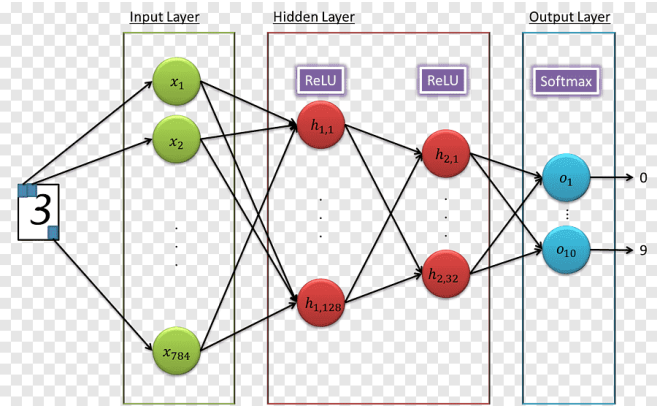

To make things easier we are going to add two hidden layers each one with 64 neurons as our default value, any way the user will be available to change this value to check the different results depending on:

- Number of hidden layers
- Number of neurons on each layer

For our Output layer we are going to use 10 Neurons representing the numbers, 0,1,2,3,4,5,6,7,8,9 each one.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

TRAINING MODEL:




On the last page we design how our layers will be working, meaning that our program will take the decision of which number is, any way at the beginning our program will be very imprecise, since by not having any reference, we will not be able to have an acceptable answer, that is why we have to train our program to be able to recognize the numbers.

As we have told we are going to use MNSIT Dataset which have more lest more than 60.000 images for training and more than 10.000 for testing.

But how we are going to train it?

The answer is simple "**COST FUNCTION**", understanding this concept is very easy , let's suppose that we give our program N number of numbers and the program will return what it things it represent, of course the first times will be wrong, but there is when the cost function starts, the function will tell de program what should have being and the program will "learn" from it, this we will repeat it a lot of times.

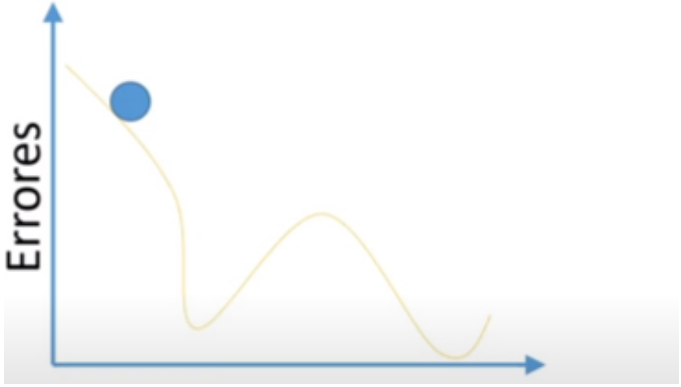
Example of cost function on the result of 1:16

**Difference:** 5

**Difference:** 5

**Difference:** 6

**Total errors:** 16

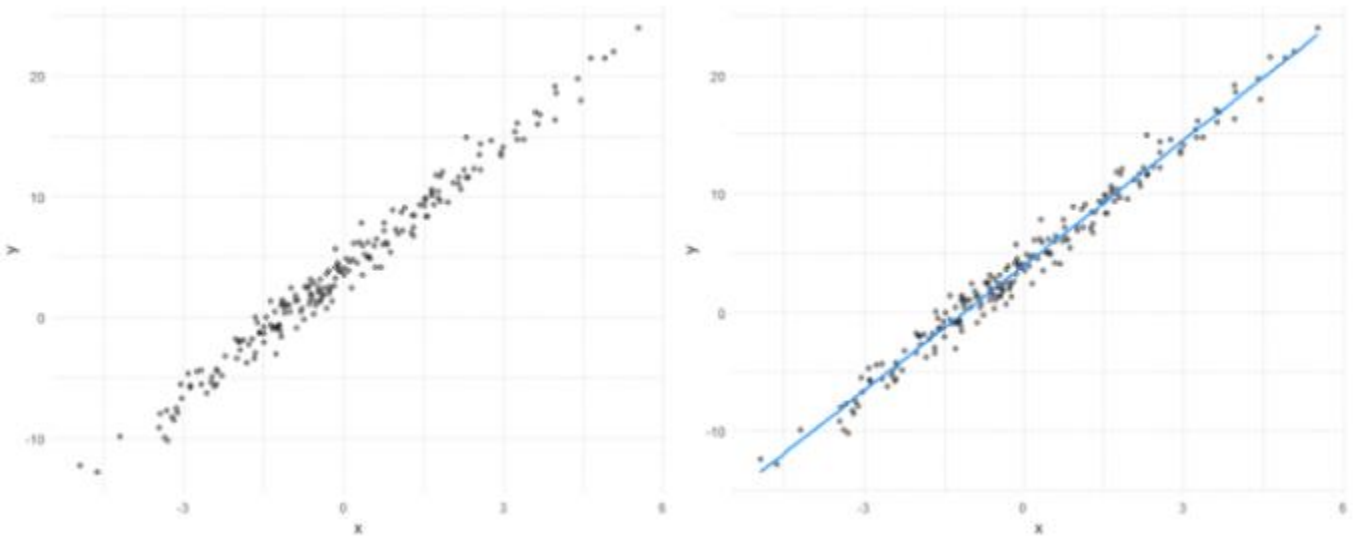


This means that we want to minimize the errors (represented with the blue dot) and getting the least errors (lower points on the chart) resulting on least error for our program.

Know the next step is to optimize this result, meaning that the next time will adjust the neurons so the program will give us more exact results.

How dose the function cost works?

Cost functions are used to estimate how badly models are performing. , *"A cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between X and y."*



How does the Optimizer work?

It’s mostly impossible to know what our model’s weights should be right from the start. But with some trial and error based on the loss function we can end up getting there in some moment.

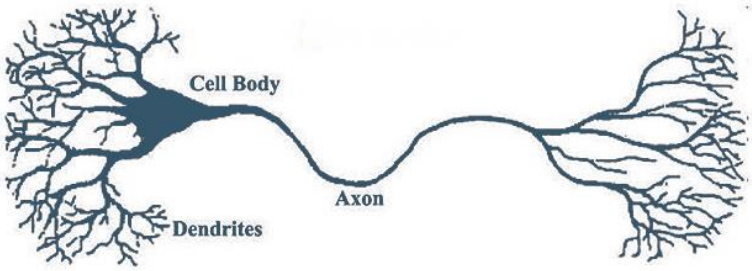
How you should change your weights or learning rates of your neural network to reduce the losses is defined by the optimizers you use. Optimization algorithms are responsible for reducing the losses and to provide the most accurate results possible.

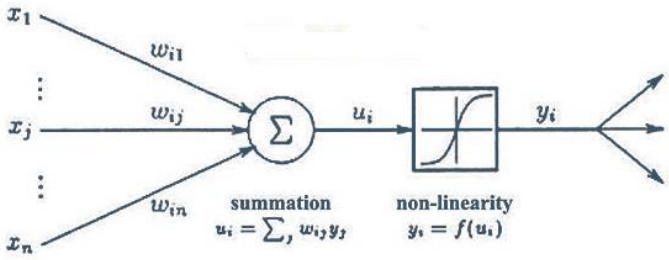
There are different ways to optimize.

- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Mini Batch Stochastic Gradient Descent (MB-SGD)
- SGD with momentum
- Nesterov Accelerated Gradient (NAG)
- Adaptive Gradient (AdaGrad)
- AdaDelta
- RMSprop
- Adam

On this project we use the adam optimizer to take the metrics, any way we can choose between some of these [ https://www.tensorflow.org/api_docs/python/tf/keras/optimizers]

- Adadelta: Optimizer that implements the Adadelta algorithm.
- Adagrad: Optimizer that implements the Adagrad algorithm.
- Adam: Optimizer that implements the Adam algorithm.
- Adamax: Optimizer that implements the Adamax algorithm.
- Ftrl: Optimizer that implements the FTRL algorithm.
- Nadam: Optimizer that implements the NAdam algorithm.
- Optimizer: Base class for Keras optimizers.
- RMSprop: Optimizer that implements the RMSprop algorithm.
- SGD: Gradient descent (with momentum) optimizer.




## Weights in connections:

We give it the name of connection weight when we talk about the number that represents how much relationship one neuron has with another, which makes some neurons have a stronger relationship than others.

On this case we are going to have more than 50.000 connections that can be adjusted.

## Neuron thresholds:

This is the name given to the number that each neuron has and is used to decide how each one works. On this case we have more that 100

We can imagine that each time the neuron will be more specialized to identify a specific path, like curves or positions, this will mean that some of the threshold will have more weight to target a specific output.

Summarizing, on this project we are touching this topic:

1. Dense layers.
2. Deep network
3. Supervised learning
4. Batch learning
5. Classification

**Dense Layers:** Each of the neurons in the anterior layer connects with one of the neurons in the next layer.

**Deep network:** we are using more than one hidden layer.

**Supervised Learning:** By training our program with images where we already know which is the correct answer.

**Batch learning:** We measure the errors with the cost function, and the optimizer adjusts the weights and thresholds.

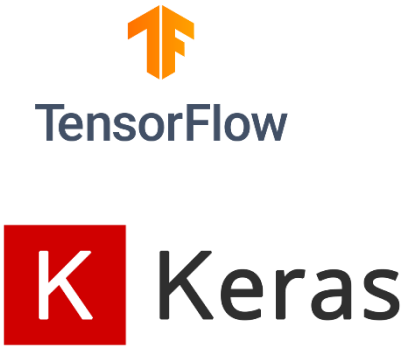
**Classification:** We always want the program to decide and classify the images in a number from 0 to 9

TENSORFLOW + KERAS:

For the implementation we are going to use python and TensorFlow + Keras framework

First what is keras and TensorFlow?

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive and flexible ecosystem of tools, libraries, and community resources that enables researchers to innovate with machine learning and Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs,




See more on:
[https://keras.io/]

[https://www.tensorflow.org/]

We will see deeper the use of the code on the section “CODE” for the moment we are going to explain how we are using the framework.

Firstly we are managing to use the loggers and import the MNIST datasets form the TensorFlow libraries, for the Layer initialization and compiler.

Logger

[https://www.tensorflow.org/api_docs/python/tf/get_logger]

Model layer

[https://www.tensorflow.org/api_docs/python/tf/keras/Sequential]

No further we are using keras for the dense layer initialization, that way we can manage to play with those values to experiment with different results.

[https://www.tensorflow.org/api_docs/python/tf/keras/layers]

As we have seen we can use different type of optimizers, any way we are using Adam for the testing

CODE:

The first sept is getting python from 3.5 to 3.7 (if we use 3.8 or more, will fail for version control on keras framework) and install all the libraries that we need you can see more about this on the README.MD

1.- We are going to add the general imports and the import for the keras and MNSIT sets

2.- We use the dataset and initialize with the loggers form tenser flow

```
from __future__ import absolute_import, division, print_function, unicode_literals
# TensorFlow and KERAS Framework
import tensorflow_datasets as tfds
from libs import *

# Basic Libraries
import logging
import matplotlib.pyplot as plt
import math

# Start initital loggers of tenserflow/keras
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

3.- We get the 60.000 images as the training set and 10.000 for the testing

```
# We import the dataset of MNIST from the
dataset, metadata = tfds.load('mnist', as_supervised=True, with_info=True)
datasetTesting, datasetTest = dataset['train'], dataset['test']
```

4.- We are going to need a classification name for each number, so on the libs.py we add it

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

numberNames = [
    'Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six',
    'Seven', 'Eight', 'Nine'
]
```

5.- We got the examples on variables this are the 60.000 images, so we split it on 60.000 different variables

```
# Initializaction of test numbers
number_train = metadata.splits['train'].num_examples # 60.000 Data
number_examples = metadata.splits['test'].num_examples # 10.000 Data
```

6.- we use a function to split the images on pixels on this case the function will be named

```
# split the pixel numbers into black or winte (255 to 0 or 1)
def imageWitheBlak(images, labels):
    images = tf.cast(images, tf.float32)
    images /= 255
    return images, labels
```

7.- we call the function on each variable of the dataset (step 5)

8.- We need to define the network structure, so we add it to the model with KERAS

```
# General initiation of the neural Network with the number of layers
model = tf.keras.Sequential([

    # Start the model with the number of nerual networks (784)
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),# 28 X 28

    # Split two new 64 size Dense layaers
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),

    # Output Layer
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  # Clasiffication
])
```

We are using “Relu” for the dense layers and SoftMax for the classification output layer.

9.-We compile the model using specific optimizers and loss categories

```
# Compile function to use
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

10.- Now we need to configure our set, with a batch of 32 (but the user can manage to change it later, and we randomize the training dataset

```
# Learning rate into 32 blocks
size = 32
datasetTesting = datasetTesting.repeat().shuffle(number_train).batch(size)
datasetTest = datasetTest.batch(size)
```

11.- Next we need to specify how many loops or epochs we are going to manage so our model can learn and we evaluate the process.

```
# Start the Model learning process
model.fit(
    datasetTesting, epochs=3,  # Number of epocs to use
    steps_per_epoch=math.ceil(number_train / size)
)

# Evaluation of the Model
loss, accuracy = model.evaluate(
    datasetTest, steps=math.ceil(number_examples / 32)
)
```

12.- Finally so we can manage to see the results we will use the Plot Images and Plot Graph to visualize the resulting classification.

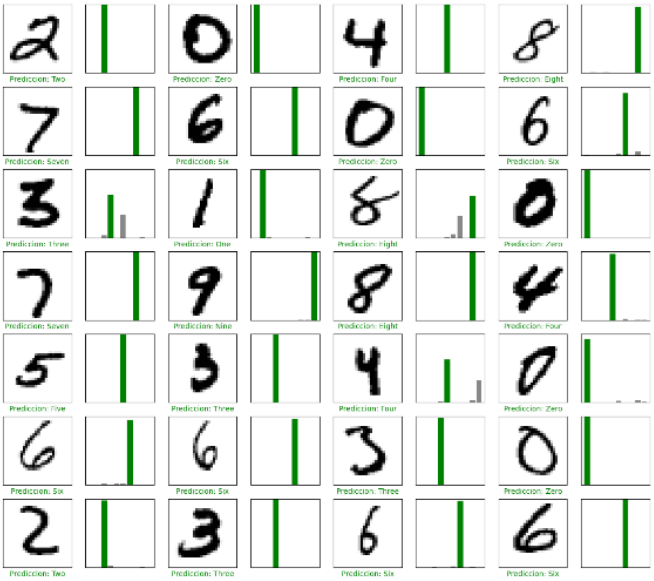
```
def plotImages(i, predictions, finalLabel):
    predictions, finalLabel = predictions[i], finalLabel[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions, color="#888888")
    plt.ylim([0, 1])
    finalLabels = np.argmax(predictions)

    thisplot[finalLabels].set_color('red')
    thisplot[finalLabel].set_color('blue')


def plotGraph(i, predictions, finalLabels, images):
    predictions, finalLabel, img = predictions[i], finalLabels[i], images[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img[..., 0], cmap=plt.cm.binary)

    finalLabels = np.argmax(predictions)
    if finalLabels == finalLabel:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("Prediccion: {}".format(numberNames[finalLabels]), color=color)
```

RESULTS

On the first evaluations we use only 1 Epoch and a network of 3 neurons and a batch of 16

This is the first a}with the minimum, and as we can see on mostly all the cases the program doubt on the number that its analyzing. For example on the number 6 it was a fight between zero and six but also a 2 o, we can look for many other examples where it fails for a minimum but if fail, like the 0 , where its confused for a six

On the next wave we test it with 3 epochs, 64 neurons and a batch of 32



We can see that is actually more accurate , but there are some images that doubts a little, like with the eight, any way the percentage of accuracy is 94% a very good result

If you want to see all the testing you can look for the TESTS.xlx on the documentation carpet

REFERENCES:

KAGGEL. (2015). Digit Recognizer. 18/04/2021, de KAGGEL Sitio web: https://www.kaggle.com/c/digit-recognizer

Conor .M. (2017). Machine learning fundamentals (I): Cost functions and gradient descent. 19/04/2021, de towardsdatascience Sitio web: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220#:~:text=Put%20simply%2C%20a%20cost%20function,value%20and%20the%20actual%20value.

KDNuggets. (2018). Optimization Algorithms in Neural Networks. 19/04/2021, de KDNuggets Sitio web: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html#:~:text=Optimizers%20are%20algorithms%20or%20methods,problems%20by%20minimizing%20the%20function.

Ellie Birbeck. (2018). How To Build a Neural Network to Recognize Handwritten Digits with TensorFlow. 10/04/2021, de Digital Ocean Sitio web: https://www.digitalocean.com/community/tutorials/how-to-build-a-neural-network-to-recognize-handwritten-digits-with-tensorflow