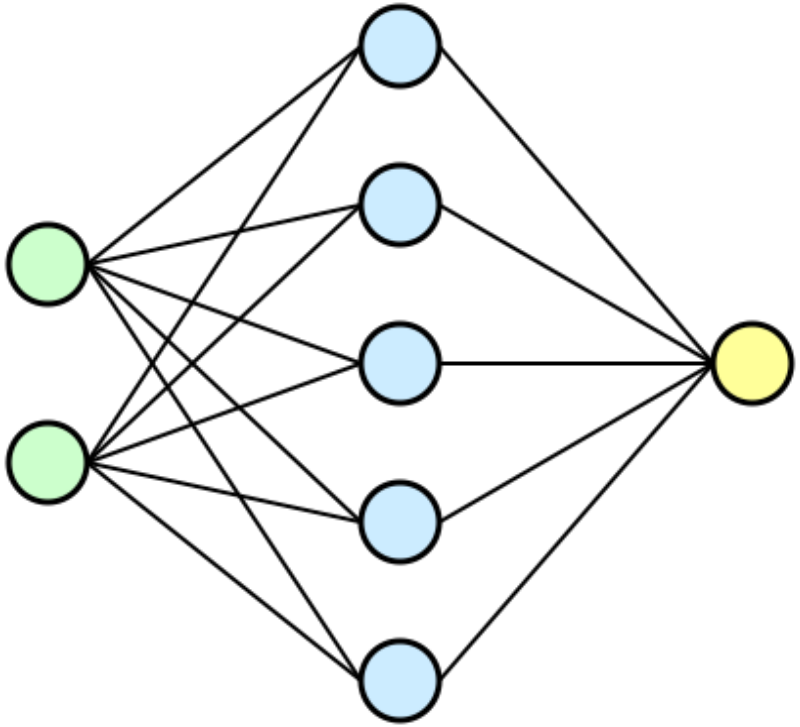


COLIN BERRY
CHRISTOPHER CATRON
JAKE HORIO

HANDWRITTEN DIGIT RECOGNITION

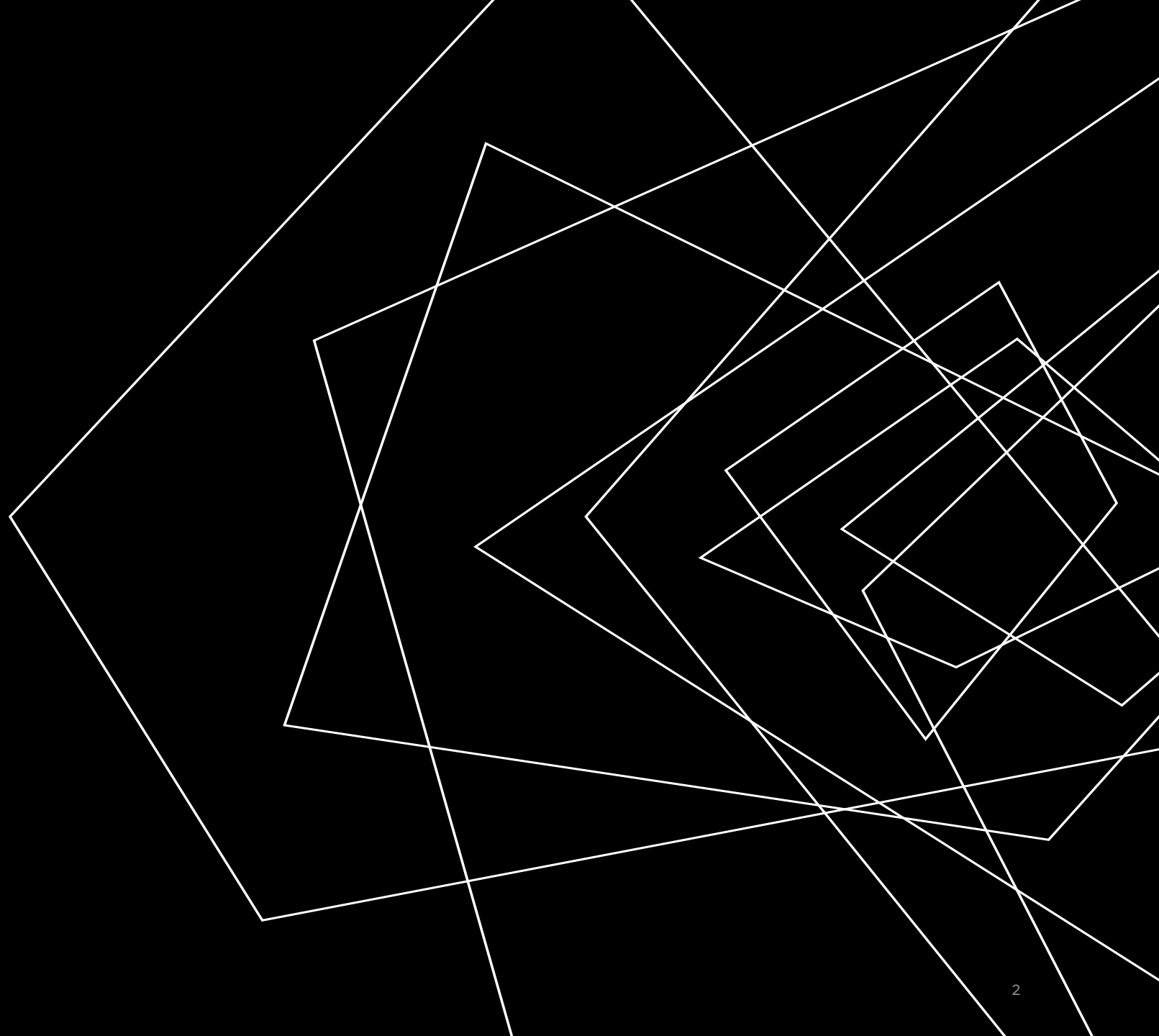
NEURAL NETWORKS IN DATA SCIENCE

AGENDA

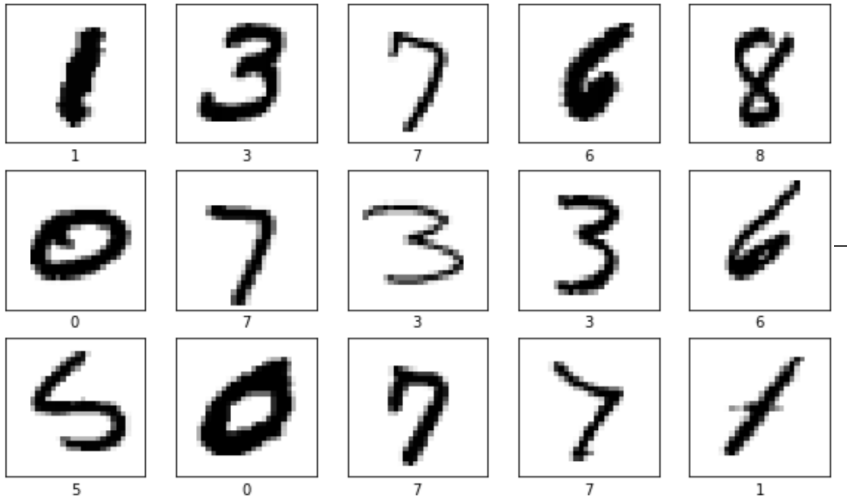
Introduction / Theory

Building Network

Demo



INTRODUCTION / THEORY



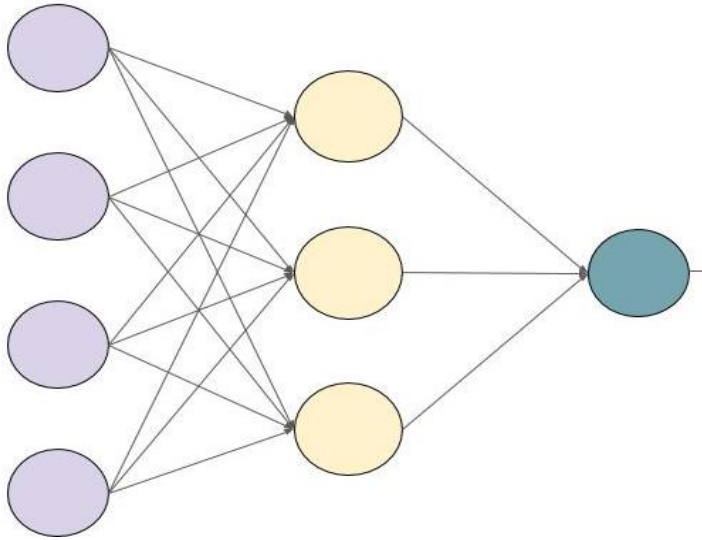
INTRODUCTION / THEORY [1]

Goal of Project

- To create a fully connected, multi-layer perceptron neural network that can recognize handwritten signatures using the MNIST dataset without using high-level libraries to implement the network architecture, showcasing our in-class knowledge.

The MNIST Dataset

- The MNIST database contains handwritten digits with a training set of 60,000 examples and a test set of 10,000 examples.
- Each image is a grayscale image of size 28x28 pixels.
- Each pixel has a pixel value ranging from 0 (black) - 255 (white). We divided each pixel value by 255 to get values that the network can work with (0-1).
- We transpose the matrix for each image so that columns represent each image and rows (784 total) represent the pixels of that image.



INTRODUCTION / THEORY [2]

Network Architecture

- This project uses a three-layered neural network, which consists of an input layer, a hidden layer, and an output layer.
- The input layer will receive initial data and pass it to hidden layer, representing raw input data in a format that the network can understand.
- The hidden layer extracts features from the input data through weighted calculation –including bias – and activation functions.
- The output layer will produce the final predictions. This project's output layer will contain 10 neurons for the 0-9 output possibilities. The values of each neuron will be a value 0-1. As a result, we needed to one-hot encode the labels, which represent the number of each image in MNIST dataset.

INTRODUCTION / THEORY [3]

Input Layer

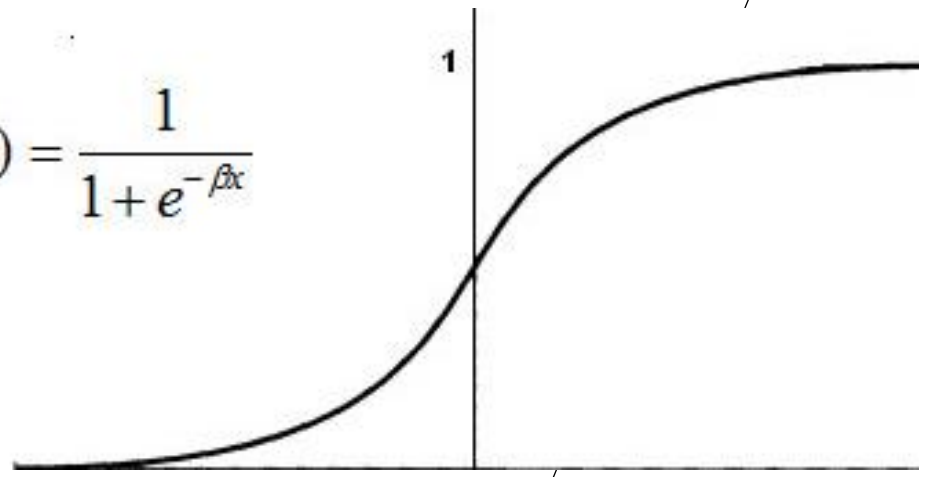
- Since the MNIST images are 28x28 pixels, our input layer will contain 784 (28x28) neurons to receive a single vector of pixel values.
- Each input neuron will represent a single pixel value and the activation value will be the intensity of the pixel from a 0-255 [0-1].
- A bias neuron with an activation value of 1 to shift the output.

INTRODUCTION / THEORY [4]

Hidden Layer

- Our network will contain 50 hidden layer neurons (excluding bias)
- Each neuron will apply an activation function (we used the sigmoid function) to the weighted sum of inputs (dot product)
- Like the input layer, we added a bias neuron with an activation value of 1 to shift the activation function's output.

$$f(x) = \frac{1}{1+e^{-\beta x}}$$



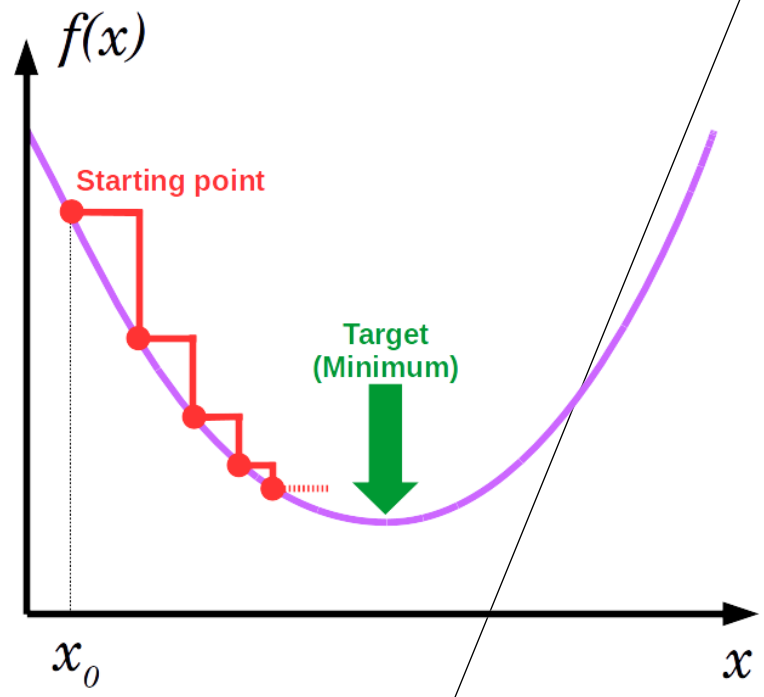
INTRODUCTION / THEORY [5]

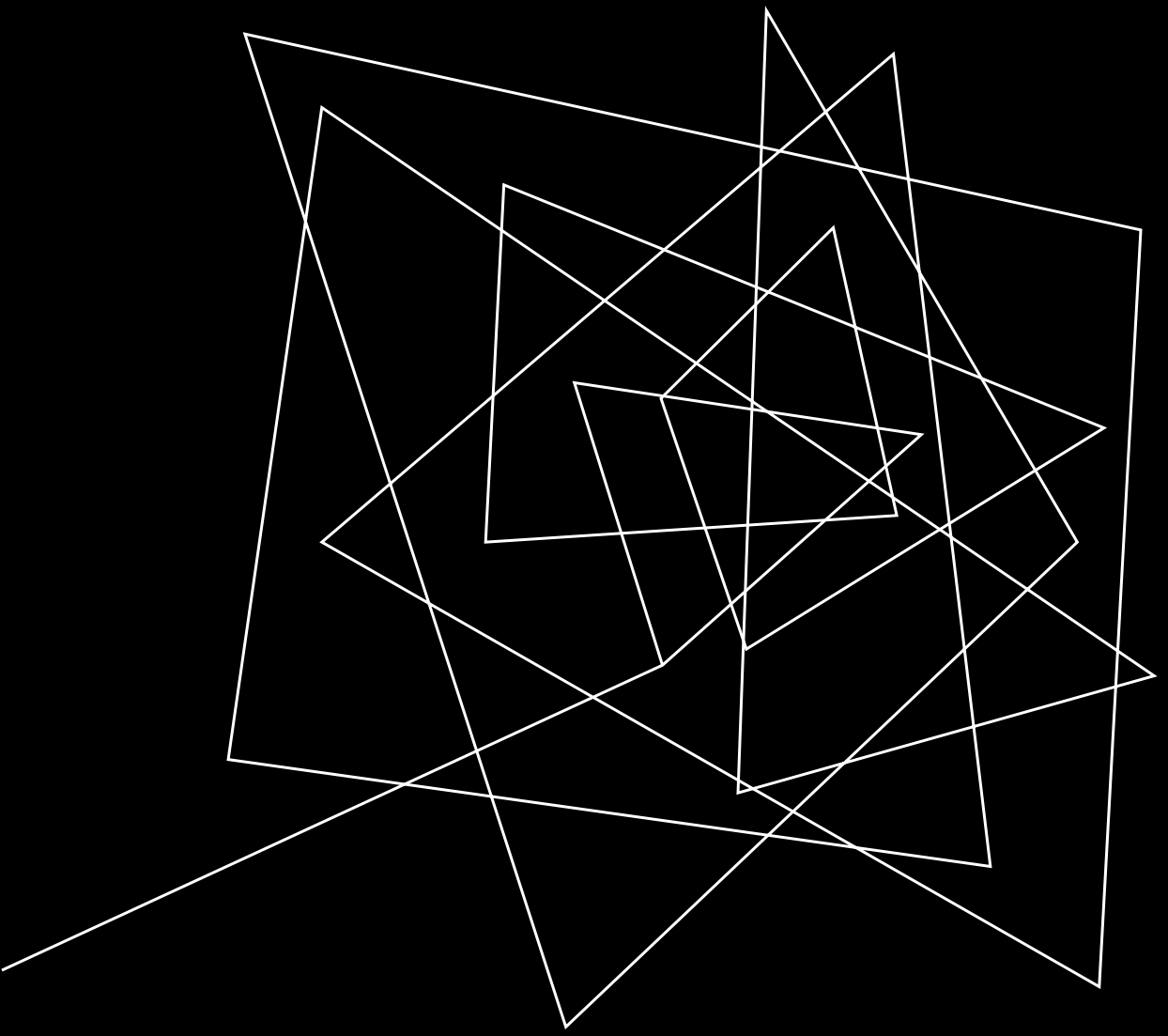
Forward Propagation

- Feedforward is performed using the MNIST training set to calculate predicted outputs.
- It is used to make predictions on unseen input data once ready for testing.

Backpropagation

- Backpropagation is used to reduce the error between layers via gradient descent (calculated using chain rule). We used the BFGS algorithm for optimization.





BUILDING THE NETWORK

FIRST STEPS

Downloading the MNSIT dataset

We downloaded the MNIST dataset and placed the data in a .mat file to be read form.

Installing and Importing Python Libraries

***NumPy*:** Used for math functions and matrix operations

***SciPy*:** Used to load .mat files into our Python program and provide optimization functions

Python Imaging Library (PIL): Used to capture image via screenshot of digit from GUI

Tkinter: Used to create the GUI for drawing the digits via cursor

```
def clear_canvas():
    canvas.delete("all")
    prediction_label.config(text="")  # Clear prediction label


# Function to handle canvas event activation
def event_activation(event):
    global lastx, lasty
    canvas.bind('<B1-Motion>', draw_lines)
    lastx, lasty = event.x, event.y


# Function to draw lines on canvas
def draw_lines(event):
    global lastx, lasty
    x, y = event.x, event.y
    canvas.create_line((lastx, lasty, x, y), width=40, fill='white', capstyle=ROUND, smooth=TRUE, splinesteps=2)
    lastx, lasty = x, y


# Create canvas
canvas = Canvas(window, width=400, height=400, bg='black')
canvas.place(x=10, y=10)
canvas.bind('<Button-1>', event_activation)

# Create buttons
clear_button = Button(window, text="Clear Canvas", font=('Ubuntu', 15), bg="blue", fg="white", command=clear_canvas)
clear_button.place(x=140, y=430)

predict_button = Button(window, text="Recognize", font=('Ubuntu', 15), bg="blue", fg="white", command=calculate_prediction)
predict_button.place(x=500, y=430)

# Label for prediction result
prediction_label = Label(window, font=('Ubuntu', 130), fg="blue")
prediction_label.place(x=510, y=120)

lastx, lasty = None, None
window.geometry("700x490")
window.mainloop()
```

```
At iterate  113    f=  1.16761D-01    |proj g|=  8.90929D-04

At iterate  114    f=  1.15914D-01    |proj g|=  1.48222D-03

At iterate  115    f=  1.14303D-01    |proj g|=  6.47944D-04

At iterate  116    f=  1.12964D-01    |proj g|=  3.72325D-04

At iterate  117    f=  1.11969D-01    |proj g|=  1.21267D-03
```

TENSORFLOW AND CNNS

Tensorflow:

- Python library that makes it very easy to build a neural network. Keras is a high-level library built on top of TF that makes it even easier.

Convolutional Neural Network (CNN)

- Feed-forward network that uses convolutional layers to learn hierarchical representations of spatial features from input images via the use of filters/kernels to detect complicated patterns.
- This use of filters and convolving for pattern recognition is ideal for image classification and object detection.

Using **Tensorflow/Keras**, we also created a **CNN** to recognize handwritten digits to showcase Tensorflow's impressive capabilities and the benefits of using a CNN for **image classification.**

REFERENCES

GfG. (2021, October 29). *Handwritten digit recognition using neural network*. GeeksforGeeks. https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/