Untitled-2

```python
# %% [markdown]
# # ***`1`***

# %%
import numpy as np

def hebbian(x, y, w, lr=0.1):
    return w + lr * np.outer(x, y)

def perceptron(x, y, w, lr=0.1):
    return w + lr * x * y

def delta(x, y, w, lr=0.1):
    return w + lr * (y - np.dot(w, x)) * x

def correlation(x, y, w, lr=0.1):
    return w + lr * np.outer(x, y)

def outstar(x, y, w, lr=0.1):
    return w + lr * (y - np.dot(w, x))

# Example usage
x = np.array([1, -1, 0, 0.5])
y = 1
w = np.array([0.2, -0.1, 0.0, 0.1])

w_hebbian = hebbian(x, y, w)
w_perceptron = perceptron(x, y, w)
w_delta = delta(x, y, w)
w_correlation = correlation(x, y, w)
w_outstar = outstar(x, y, w)

print("Hebbian:", w_hebbian)
print("Perceptron:", w_perceptron)
print("Delta:", w_delta)
print("Correlation:", w_correlation)
print("OutStar:", w_outstar)

# %% [markdown]
# # ***`2`***

# %%
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
plt.plot(x, 1 / (1 + np.exp(-x)), label='Sigmoid')
plt.plot(x, np.tanh(x), label='Tanh')
plt.plot(x, np.maximum(0, x), label='ReLU')
plt.plot(x, np.where(x > 0, x, x * 0.01), label='Leaky ReLU')
plt.plot(x, np.exp(x) / np.sum(np.exp(x)), label='Softmax')
plt.legend()
plt.show()


# %% [markdown]
# ## ***`3`***

# %%
import numpy as np

# Define the inputs and weights
inputs = np.array([
    [1, 1, 1],  # Favorite hero, heroine, Climate (all conditions met)
    [1, 0, 1],  # Favorite hero, not heroine, Climate
    [0, 1, 1],  # Not Favorite hero, heroine, Climate
    [0, 0, 1],  # Not Favorite hero, not heroine, Climate
    [1, 1, 0],  # Favorite hero, heroine, not Climate
    [1, 0, 0],  # Favorite hero, not heroine, not Climate
    [0, 1, 0],  # Not Favorite hero, heroine, not Climate
    [0, 0, 0]   # Not Favorite hero, not heroine, not Climate
])

# Target outputs (1 = go to movie, 0 = don't go)
targets = np.array([1, 0, 1, 0, 0, 0, 0, 0])

# Weights and bias
weights = np.array([0.2, 0.4, 0.2])
bias = -0.5

# Perceptron function
def perceptron(inputs, weights, bias):
    output = np.dot(inputs, weights) + bias
    return np.where(output > 0, 1, 0)

# Predictions
predictions = perceptron(inputs, weights, bias)

# Calculate accuracy
accuracy = np.mean(predictions == targets)
print(f"Predictions: {predictions}")
print(f"Accuracy: {accuracy * 100}%")

# Output results
for i, input_case in enumerate(inputs):
    decision = "Go to movie" if predictions[i] == 1 else "Don't go to movie"
    print(f"Input: {input_case}, Decision: {decision}")


# %% [markdown]
# ### ***`4`***

# %%
import cv2
import numpy as np
```

```python

def process_image(image_path):
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Apply histogram equalization
    equalized = cv2.equalizeHist(img)

    # Apply binary thresholding
    _, thresholded = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)

    # Detect edges using Canny
    edges = cv2.Canny(img, 100, 200)

    # Flip the image horizontally
    flipped = cv2.flip(img, 1)

    # Apply morphological closing operation
    kernel = np.ones((5, 5), np.uint8)
    morphed = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

    # Display the results
    cv2.imshow('Original', img)
    cv2.imshow('Equalized', equalized)
    cv2.imshow('Thresholded', thresholded)
    cv2.imshow('Edges', edges)
    cv2.imshow('Flipped', flipped)
    cv2.imshow('Morphed', morphed)

    # Wait for a key press and close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Make sure to replace the image_path with your actual image file path
image_path = r'C:\Users\crick\OneDrive\Desktop\HD-wallpaper-evening-pic-natura-thumbnail.jpg'
process_image(image_path)


# %% [markdown]
# ### ***`5`***
#
#

# %%
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt

def load_and_process_image(image_path):
    img = tf.io.read_file(image_path)  # Read the image file
    img = tf.image.decode_image(img, channels=3)  # Decode the image as a 3-channel (RGB) image
    img = tf.image.resize(img, [512, 512])  # Resize the image to 512x512 pixels
    img = img / 255.0  # Normalize the pixel values to [0, 1]
    return img[tf.newaxis, ...]  # Add a batch dimension

# Load the pre-trained model from TensorFlow Hub
model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')

# Load and process the content and style images
content_image = load_and_process_image('/home/lab705/Downloads/1.jpeg')
style_image = load_and_process_image('/home/lab705/Downloads/2.jpeg')

# Perform style transfer
stylized_image = model(content_image, style_image)[0]

# Plot the content image, style image, and stylized image
plt.figure(figsize=(12, 4))
for i, img in enumerate([content_image, style_image, stylized_image]):
    plt.subplot(1, 3, i+1)
    plt.imshow(img[0])
    plt.axis('off')
plt.show()


# %% [markdown]
# ### ***`6`***

# %%
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
import numpy as np
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define class names
class_names = ["Airplane", "Automobile", "Bird", "Cat", "Deer", "Dog", "Frog", "Horse", "Ship", "Truck"]

# Create a simple model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Make a prediction
```

```python
test_image=x_test[10]
prediction = model.predict(test_image[np.newaxis, ...])
predicted_class = np.argmax(prediction)
print(predicted_class)

# Display the image and prediction
plt.imshow(test_image)
plt.title(f"Predicted: {class_names[predicted_class]}")
plt.axis('off')
plt.show()

print(f"Predicted class: {class_names[predicted_class]}")

# %% [markdown]
# ### ***`7`***

# %%
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1)) / 255.0
test_images = test_images.reshape((10000, 28, 28, 1)) / 255.0

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
image=test_images[10]
prediction = model.predict(image[np.newaxis, ...])
predicted_label = np.argmax(prediction)
print(predicted_label)
plt.imshow(image)


# %% [markdown]
# ### ***`8`***

# %%
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Load the dataset
data = pd.read_csv(r'C:\Users\crick\OneDrive\Desktop\sonar.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(x_train.shape)

# Define the model architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Test accuracy:', test_acc)

# Make predictions on the test set
predictions = model.predict(X_test)
predictions = np.where(predictions > 0.5, 1, 0)
print(predictions)


# %%
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```
import tensorflow as tf

# Load and prepare data
data = pd.read_csv(r'C:\Users\crick\OneDrive\Desktop\sonar.csv')
X = StandardScaler().fit_transform(data.iloc[:, :-1].values)
y = LabelEncoder().fit_transform(data.iloc[:, -1].values)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build and compile model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(60, activation='relu', input_dim=60),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(30, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train and evaluate model
model.fit(X_train, y_train, epochs=50, validation_split=0.1)
print("Test Accuracy:", model.evaluate(X_test, y_test)[1])


# %% [markdown]
# ### ***`9`***

# %%
import tensorflow as tf
from tensorflow.keras import layers, Sequential
import matplotlib.pyplot as plt
import numpy as np

def make_generator():
    return Sequential([
        layers.Dense(7*7*256, use_bias=False, input_shape=(100,)),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Reshape((7, 7, 256)),
        layers.Conv2DTranspose(128, 5, strides=1, padding='same', use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Conv2DTranspose(64, 5, strides=2, padding='same', use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Conv2DTranspose(1, 5, strides=2, padding='same', use_bias=False, activation='tanh')
    ])

def make_discriminator():
    return Sequential([
        layers.Conv2D(64, 5, strides=2, padding='same', input_shape=(28, 28, 1)),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Conv2D(128, 5, strides=2, padding='same'),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1)
    ])

generator = make_generator()
discriminator = make_discriminator()

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
gen_opt = tf.keras.optimizers.Adam(1e-4)
disc_opt = tf.keras.optimizers.Adam(1e-4)

@tf.function
def train_step(images, batch_size=32):
    noise = tf.random.normal([batch_size, 100])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = cross_entropy(tf.ones_like(fake_output), fake_output)
        disc_loss = cross_entropy(tf.ones_like(real_output), real_output) + cross_entropy(tf.zeros_like(fake_output), fake_output)
    generator_gradients = gen_tape.gradient(gen_loss, generator.trainable_variables)
    discriminator_gradients = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    gen_opt.apply_gradients(zip(generator_gradients, generator.trainable_variables))
    disc_opt.apply_gradients(zip(discriminator_gradients, discriminator.trainable_variables))

# Load and preprocess the MNIST dataset
(train_images, _), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5  # Normalize to [-1, 1]

batch_size = 256
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(60000).batch(batch_size)

# Training loop
epochs = 50
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, 100])

def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, training=False)
    fig = plt.figure(figsize=(4, 4))

    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow((predictions[i, :, :, 0] + 1) / 2, cmap='gray')
        plt.axis('off')

    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
    plt.show()

for epoch in range(epochs):
    for image_batch in train_dataset:
        train_step(image_batch)

    generate_and_save_images(generator, epoch + 1, seed)

generate_and_save_images(generator, epochs, seed)


# %% [markdown]
# ### ***`10`***

# %%
import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, Flatten
from tensorflow.keras.models import Sequential


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

def train_model(use_batch_norm=False):
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu' if not use_batch_norm else None),
        (BatchNormalization() if use_batch_norm else Dropout(0.2)),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    model.fit(x_train, y_train, epochs=20, verbose=0)
    return model.evaluate(x_test, y_test, verbose=0)

print("Dropout Model Accuracy:", train_model(use_batch_norm=False)[1])
print("Batch Norm Model Accuracy:", train_model(use_batch_norm=True)[1])


# %% [markdown]
# ### ***`11`***

# %%
!pip install tf-models-official
!pip install tensorflow


# %%
!pip install tensorflow
!pip install opencv-python
!pip install numpy==1.19.5  # Install a specific version (replace 1.19.5 with a compatible version)
```

```

# %%
# Clone the Mask R-CNN repository if not already present
!git clone https://github.com/akTwelve/Mask_RCNN.git

import os
import sys
import skimage.io
import matplotlib.pyplot as plt
import tensorflow as tf

# Set up paths and configurations
ROOT_DIR = "Mask_RCNN"
MODEL_DIR = os.path.join(ROOT_DIR, "logs")
IMAGE_PATH = "/content/abcd.png"
COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

# Import Mask R-CNN and COCO configuration
sys.path.append(ROOT_DIR)
from mrcnn import utils, model as modellib, visualize
sys.path.append(os.path.join(ROOT_DIR, "samples/coco/"))
import coco

# Download COCO weights if not already present
if not os.path.exists(COCO_MODEL_PATH):
    utils.download_trained_weights(COCO_MODEL_PATH)

# Inference configuration for CPU
class InferenceConfig(coco.CocoConfig):
    GPU_COUNT = 0
    IMAGES_PER_GPU = 1

config = InferenceConfig()
model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=config)
model.load_weights(COCO_MODEL_PATH, by_name=True)

# Load image
image = skimage.io.imread(IMAGE_PATH)

# Perform detection
results = model.detect([image], verbose=0)
r = results[0]

# Class names for COCO dataset
class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
               'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign',
               'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep',
               'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella',
               'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard',
               'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
               'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange',
               'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair',
               'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv',
               'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave',
               'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase',
               'scissors', 'teddy bear', 'hair drier', 'toothbrush']

# Visualize results
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'], class_names, r['scores'])



# %% [markdown]
# ### ***`12`***

# %%
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load a small sample text corpus (replace with your own)
corpus = [
    "Hello, how are you?",
    "I am doing well.",
    "What's your name?",
    "My name is ChatBot.",
    "How can I help you?",
    "Tell me a joke.",
    "Why did the chicken cross the road?",
    "To get to the other side."
]

# Tokenization and Vocabulary Creation
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
vocab_size = len(tokenizer.word_index) + 1  # Add 1 for padding token
word_index = tokenizer.word_index

# Set model parameters
embedding_dim = 128
lstm_units = 64
max_length = 10  # Adjust based on your corpus

# Prepare training data
sequences = tokenizer.texts_to_sequences(corpus)
padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='pre')

# Split data into input and target sequences
input_sequences = padded_sequences[:, :-1]
target_sequences = padded_sequences[:, 1:]

# Model Definition
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length-1),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(lstm_units, return_sequences=True)),
    tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(vocab_size, activation='softmax'))
])

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

# Reshape target data to match output shape of the model
target_sequences = target_sequences.reshape((target_sequences.shape[0], target_sequences.shape[1], 1))

# Train the model (adjust epochs based on your data)
model.fit(input_sequences, target_sequences, epochs=200)

def generate_response(input_query, max_length=10):
    input_sequence = tokenizer.texts_to_sequences([input_query])[0]
    input_sequence = pad_sequences([input_sequence], maxlen=max_length-1, padding='pre')

    predicted_sequence = model.predict(input_sequence)
    predicted_words = [np.argmax(predicted_sequence[0][i]) for i in range(predicted_sequence.shape[1])]
    response = tokenizer.sequences_to_texts([predicted_words])[0]
    return response

# Example interaction
user_query = "Hello, how are you?"
response = generate_response(user_query)
print(f"User: {user_query}")
print(f"ChatBot: {response}")


# %% [markdown]
# ### ***`13`***

# %%
import tensorflow as tf
import cv2
import numpy as np

model = tf.keras.models.load_model('yolo.h5')
classes = ["person", "bicycle", "car", "..."]

image = cv2.imread('coco_image.jpg')
input_image = cv2.resize(image, (416, 416))
input_image = input_image / 255.0
input_image = np.expand_dims(input_image, 0)

detections = model.predict(input_image)

boxes = detections[:, :, :4]
scores = detections[:, :, 4]
class_ids = np.argmax(detections[:, :, 5:], axis=-1)


for i in range(len(boxes)):
    if scores[i] > 0.5:  # Confidence threshold
        x, y, w, h = boxes[i]
        x, y, w, h = int(x), int(y), int(w), int(h)
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        label = classes[class_ids[i]]
        cv2.putText(image, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

cv2.imshow("Detections", image)
cv2.waitKey(0)

# %% [markdown]
# ### ***`14`***

# %%
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence

# Load IMDB dataset (using top 5000 most frequent words)
vocab_size = 5000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Pad sequences to have the same length
max_len = 500
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)

# Build GRU model
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(vocab_size, 128),
    tf.keras.layers.GRU(128),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=64)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print("Accuracy:", accuracy)

# %% [markdown]
# ### ***`15`***

# %%

import torch
from transformers import T5ForConditionalGeneration, T5Tokenizer

model_name = "t5-small"
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

```

```python
def summarize_text(text, max_length=50):
    input_text = "summarize: " + text
    input_ids = tokenizer.encode(input_text, return_tensors="pt", add_special_tokens=True)
    output_ids = model.generate(input_ids=input_ids, max_length=max_length, num_beams=4, no_repeat_ngram_size=2)
    summary = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    return summary

text = """
Without strong security rules, anyone who has the address of your database can read / write to it, leaving your data vulnerable to attackers stealing, modifying, or deleting data as well as creating costly operations.
"""

summary = summarize_text(text)
print(summary)
```