Sample DataSet

from sklearn.datasets import make\_circles

samples =100

X,y = make\_circles(samples,noise=0.03,random\_state=42)

import pandas as pd

circles = pd.DataFrame({"X0":X[:,0],"X1":X[:,1],"Y": y})

circles.head()

import matplotlib.pyplot as plt

plt.scatter(X[:,0],X[:,1],c=y,cmap=plt.cm.RdYlBu)

<https://playground.tensorflow.org/>

Input and Output shape

X.shape, y.shape

len(X),len(y)

X[0],y[0]

Creating a basic model

tf.random.set\_seed(42)

model = tf.keras.Sequential([

tf.keras.layers.Dense(1)

])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(),

optimizer=tf.keras.optimizers.SGD(),

metrics=["accuracy"])

model.fit(X,y,epochs=10)

model.fit(X,y,epochs=200,verbose=0)

model.evaluate(X,y)

Improving the model

CS231n - <https://cs231n.github.io/neural-networks-case-study/>: Study

2. Made with ML basics - <https://github.com/GokuMohandas/MadeWithML/blob/main/notebooks/08_Neural_Networks.ipynb>: Study

Improving our model using plot\_decision\_boundary functions

def plot\_decision\_boundary(model, X, y):

"""

Plots the decision boundary created by a model predicting on X.

This function has been adapted from two phenomenal resources:

1. CS231n - https://cs231n.github.io/neural-networks-case-study/

2. Made with ML basics - https://github.com/GokuMohandas/MadeWithML/blob/main/notebooks/08\_Neural\_Networks.ipynb

"""

# Define the axis boundaries of the plot and create a meshgrid

x\_min, x\_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1

y\_min, y\_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 100),

np.linspace(y\_min, y\_max, 100))

# Create X values (we're going to predict on all of these)

x\_in = np.c\_[xx.ravel(), yy.ravel()] # stack 2D arrays together: https://numpy.org/devdocs/reference/generated/numpy.c\_.html

# Make predictions using the trained model

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

# Check for multi-class

if len(y\_pred[0]) > 1:

print("doing multiclass classification...")

# We have to reshape our predictions to get them ready for plotting

y\_pred = np.argmax(y\_pred, axis=1).reshape(xx.shape)

else:

print("doing binary classifcation...")

y\_pred = np.round(y\_pred).reshape(xx.shape)

# Plot decision boundary

plt.contourf(xx, yy, y\_pred, cmap=plt.cm.RdYlBu, alpha=0.7)

plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

Use the classification model to predict regression data and check if its working

Classification model cant be used to predict regression data

So create a new model for regression data

tf.random.set\_seed(42)

model\_3 = tf.keras.Sequential([

tf.keras.layers.Dense(100,input\_shape=[1]),

tf.keras.layers.Dense(10),

tf.keras.layers.Dense(1)

])

model\_3.compile(loss=tf.keras.losses.mae,

optimizer=tf.keras.optimizers.Adam(),

metrics=["mae"]

)

model\_3.fit(X\_reg\_train,y\_reg\_train,epochs=100)

y\_reg\_preds = model\_3.predict(X\_reg\_test)

plt.figure(figsize=(10,7))

plt.scatter(X\_reg\_train,y\_reg\_train,c='b',label="Training Data")

plt.scatter(X\_reg\_test,y\_reg\_test,c='g',label="Testing Data")

plt.scatter(X\_reg\_test,y\_reg\_preds,c='r',label="Predictions Data")

plt.legend()

The missing non linearity

tf.random.set\_seed(42)

model\_4 = tf.keras.Sequential([

tf.keras.layers.Dense(1, activation=tf.keras.activations.linear),

tf.keras.layers.Dense(1)

])

model\_4.compile(loss=tf.keras.losses.binary\_crossentropy,

optimizer=tf.keras.optimizers.Adam(lr=0.001),

metrics=["accuracy"])

history = model\_4.fit(X,y,epochs=100)

plot\_decision\_boundary(model\_4,X,y)

Add first non linear activation method: relu

tf.random.set\_seed(42)

model\_5 = tf.keras.Sequential([

tf.keras.layers.Dense(1, activation=tf.keras.activations.relu),

tf.keras.layers.Dense(1)

])

model\_5.compile(loss=tf.keras.losses.binary\_crossentropy,

optimizer=tf.keras.optimizers.Adam(lr=0.001),

metrics=["accuracy"])

history = model\_4.fit(X,y,epochs=100)

Add More Relu layers

tf.random.set\_seed(42)

model\_6 = tf.keras.Sequential([

tf.keras.layers.Dense(4, activation=tf.keras.activations.relu),

tf.keras.layers.Dense(4, activation=tf.keras.activations.relu),

tf.keras.layers.Dense(1)

])

model\_6.compile(loss=tf.keras.losses.binary\_crossentropy,

optimizer=tf.keras.optimizers.Adam(lr=0.001),

metrics=["accuracy"])

history = model\_4.fit(X,y,epochs=100)

model\_6.evaluate(X,y)

plot\_decision\_boundary(model\_6,

X,y)

Using Sigmoid function for output layers

tf.random.set\_seed(42)

model\_7 = tf.keras.Sequential([

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_7.compile(loss=tf.keras.losses.binary\_crossentropy,

optimizer=tf.keras.optimizers.Adam(lr=0.01),

metrics=["accuracy"]

)

history = model\_7.fit(X,y,epochs=350,verbose=0)

plot\_decision\_boundary(model\_7,X,y)

Sigmoid Function

A = tf.cast(tf.range(-10,10), tf.float32)

def sigmoid(X):

return 1/(1 + tf.exp(-X))

plt.plot(A)

Relu function

def relu(X):

return tf.maximum(0,X)

relu(A)

plt.plot(relu(A))

Evaluating and optimizing with Data splitting

X\_train, y\_train = X[:80],y[:80]

X\_test, y\_test = X[80:],y[80:]

X\_train.shape,y\_train.shape,X\_test.shape,y\_test.shape

tf.random.set\_seed(42)

model\_8 = tf.keras.Sequential([

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_8.compile(loss=tf.keras.losses.BinaryCrossentropy(),

optimizer=tf.keras.optimizers.Adam(lr=0.01),

metrics=["accuracy"])

history = model\_8.fit(X\_train,y\_train,epochs=150)

model\_8.evaluate(X\_test,y\_test)

plt.figure(figsize=(12,6))

plt.subplot(1,2,1)

plt.title("Training Data")

plot\_decision\_boundary(model\_8,X\_train,y\_train)

plt.subplot(1,2,2)

plt.title("Test Data")

plot\_decision\_boundary(model\_8,X\_test,y\_test)

Plot the loss/training Curv

pd.DataFrame(history.history)

pd.DataFrame(history.history).plot()

plt.title("Model\_8 Training Curv")

Find the best learning rate through Callbacks

tf.random.set\_seed(42)

model\_9 = tf.keras.Sequential([

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_9.compile(loss=tf.keras.losses.binary\_crossentropy,

optimizer="Adam",

metrics=["accuracy"])

lr\_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch : 1e-4 \* 10\*\*(epoch/20))

history = model\_9.fit(X\_train,y\_train,epochs=100,callbacks=[lr\_scheduler])

pd.DataFrame(history.history).plot(figsize=(10,7),xlabel ="epochs")

Plot the Learning rate vs Loss

lrs = 1e-4 \* (10\*\*(np.arange(100)/20))

plt.figure(figsize=(10,7))

plt.semilogx(lrs,history.history["loss"])

plt.xlabel("Learning Rate")

plt.ylabel("Loss")

plt.title("Learning Rate vs Loss")

find the best learning rate here

Create a model with best Learning Rate and Plot Curvs

tf.random.set\_seed(42)

model\_10 = tf.keras.Sequential([

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_10.compile(loss=tf.keras.losses.BinaryCrossentropy(),

optimizer=tf.keras.optimizers.Adam(lr=0.04),

metrics=["accuracy"])

model\_10.fit(X\_train,y\_train,epochs=80)

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.title("Training Data")

plot\_decision\_boundary(model\_10,X\_train,y\_train)

plt.subplot(1,2,2)

plt.title("Testing Data")

plot\_decision\_boundary(model\_10,X\_test,y\_test)

Different Evaluation methods

Accuracy,precision, recall, F1-score, Confusion matrix

Confusion Metrix

from sklearn.metrics import confusion\_matrix

y\_preds = model\_10.predict(X\_test)

confusion\_matrix(y\_test,y\_pred)

ValueError: Found input variables with inconsistent numbers of samples: [20, 100]

#These are prediction probability due to sigmoid funtion

y\_preds

y\_test

# Round up y\_preds

tf.round(y\_preds)

import itertools

# Note: The following confusion matrix code is a remix of Scikit-Learn's

# plot\_confusion\_matrix function - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot\_confusion\_matrix.html

# and Made with ML's introductory notebook - https://github.com/GokuMohandas/MadeWithML/blob/main/notebooks/08\_Neural\_Networks.ipynb

figsize = (10, 10)

# Create the confusion matrix

cm = confusion\_matrix(y\_test, tf.round(y\_preds))

cm\_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # normalize it

n\_classes = cm.shape[0]

# Let's prettify it

fig, ax = plt.subplots(figsize=figsize)

# Create a matrix plot

cax = ax.matshow(cm, cmap=plt.cm.Blues) # https://matplotlib.org/3.2.0/api/\_as\_gen/matplotlib.axes.Axes.matshow.html

fig.colorbar(cax)

# Create classes

classes = False

if classes:

labels = classes

else:

labels = np.arange(cm.shape[0])

# Label the axes

ax.set(title="Confusion Matrix",

xlabel="Predicted label",

ylabel="True label",

xticks=np.arange(n\_classes),

yticks=np.arange(n\_classes),

xticklabels=labels,

yticklabels=labels)

# Set x-axis labels to bottom

ax.xaxis.set\_label\_position("bottom")

ax.xaxis.tick\_bottom()

# Adjust label size

ax.xaxis.label.set\_size(20)

ax.yaxis.label.set\_size(20)

ax.title.set\_size(20)

# Set threshold for different colors

threshold = (cm.max() + cm.min()) / 2.

# Plot the text on each cell

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, f"{cm[i, j]} ({cm\_norm[i, j]\*100:.1f}%)",

horizontalalignment="center",

color="white" if cm[i, j] > threshold else "black",

size=15)

confusion\_matrix(y\_test,tf.round(y\_preds))

Multiclass Classification with fashion mnist dataset

import tensorflow as tf

from tensorflow.keras.datasets import fashion\_mnist

(train\_data,train\_labels),(test\_data,test\_labels) = fashion\_mnist.load\_data()

train\_data[0].shape, train\_labels[0].shape

train\_data.shape,train\_labels.shape

train\_data[0]

import matplotlib.pyplot as plt

plt.imshow(train\_data[0])

train\_labels[0]

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# How many classes are there (this'll be our output shape)?

len(class\_names)

plot\_index = 10

plt.imshow(train\_data[plot\_index],cmap=plt.cm.binary)

plt.title(class\_names[train\_labels[plot\_index]])

Plot Random Images

import random

plt.figure(figsize=(10,7))

for i in range(4):

ax = plt.subplot(2,2,i+1)

random\_index = random.choice(range(len(train\_data)))

plt.imshow(train\_data[random\_index],cmap=plt.cm.binary)

plt.title(class\_names[train\_labels[random\_index]])

plt.axis(False)

Create Model with Flatten input and Validation in training the model

flatten\_model = tf.keras.Sequential([tf.keras.layers.Flatten(input\_shape=(28,28))])

flatten\_model.output\_shape

from sklearn.utils import validation

tf.random.set\_seed(42)

model\_11 = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28,28)),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(10,activation="softmax")

])

model\_11.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

non\_norm\_history = model\_11.fit(train\_data,train\_labels,epochs=10,validation\_data=(test\_data,test\_labels))

Use Categorical cross entropy by onehot encode the labels if those are in integers

from sklearn.utils import validation

tf.random.set\_seed(42)

model\_11 = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28,28)),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(10,activation="softmax")

])

model\_11.compile(loss=tf.keras.losses.CategoricalCrossentropy(),

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

non\_norm\_history = model\_11.fit(train\_data,tf.one\_hot(train\_labels,depth=10),epochs=10,validation\_data=(test\_data,tf.one\_hot(test\_labels,depth=10)))

tf.keras.utils.plot\_model(model\_11,show\_shapes=True)

Normalize/Scale the data for better training of the model

train\_data.min(),train\_data.max()

train\_data\_norm = train\_data/255.0

test\_data\_norm = test\_data/255.0

train\_data\_norm.min(),train\_data\_norm.max()

tf.random.set\_seed(42)

model\_12 = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28,28)),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(10,activation="softmax")

])

model\_12.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

norm\_history = model\_12.fit(train\_data\_norm,train\_labels,epochs=10,validation\_data=(test\_data\_norm,test\_labels))

Compare the normalize and non-normalize model on history

import pandas as pd

pd.DataFrame(non\_norm\_history.history).plot(title="Non normalize Data")

pd.DataFrame(norm\_history.history).plot(title="Normalize Data")

Find the best Learning Rate and train the model again

tf.random.set\_seed(42)

# Create the model

model\_13 = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)), # input layer (we had to reshape 28x28 to 784)

tf.keras.layers.Dense(4, activation="relu"),

tf.keras.layers.Dense(4, activation="relu"),

tf.keras.layers.Dense(10, activation="softmax") # output shape is 10, activation is softmax

])

# Compile the model

model\_13.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

# Create the learning rate callback

lr\_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-3 \* 10\*\*(epoch/20))

# Fit the model

find\_lr\_history = model\_13.fit(train\_data,

train\_labels,

epochs=40, # model already doing pretty good with current LR, probably don't need 100 epochs

validation\_data=(test\_data, test\_labels),

callbacks=[lr\_scheduler])

Plot curv of loss vs learning rate

# Plot the learning rate decay curve

import numpy as np

import matplotlib.pyplot as plt

lrs = 1e-3 \* (10\*\*(np.arange(40)/20))

plt.semilogx(lrs, find\_lr\_history.history["loss"]) # want the x-axis to be log-scale

plt.xlabel("Learning rate")

plt.ylabel("Loss")

plt.title("Finding the ideal learning rate");

Create model with best learning rate

# Set random seed

tf.random.set\_seed(42)

# Create the model

model\_14 = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)), # input layer (we had to reshape 28x28 to 784)

tf.keras.layers.Dense(4, activation="relu"),

tf.keras.layers.Dense(4, activation="relu"),

tf.keras.layers.Dense(10, activation="softmax") # output shape is 10, activation is softmax

])

# Compile the model

model\_14.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

optimizer=tf.keras.optimizers.Adam(lr=0.001), # ideal learning rate (same as default)

metrics=["accuracy"])

# Fit the model

history = model\_14.fit(train\_data,

train\_labels,

epochs=20,

validation\_data=(test\_data, test\_labels))

Create Confusion Matrix Function

import itertools

from sklearn.metrics import confusion\_matrix

# Our function needs a different name to sklearn's plot\_confusion\_matrix

def make\_confusion\_matrix(y\_true, y\_pred, classes=None, figsize=(10, 10), text\_size=15):

"""Makes a labelled confusion matrix comparing predictions and ground truth labels.

If classes is passed, confusion matrix will be labelled, if not, integer class values

will be used.

Args:

y\_true: Array of truth labels (must be same shape as y\_pred).

y\_pred: Array of predicted labels (must be same shape as y\_true).

classes: Array of class labels (e.g. string form). If `None`, integer labels are used.

figsize: Size of output figure (default=(10, 10)).

text\_size: Size of output figure text (default=15).

Returns:

A labelled confusion matrix plot comparing y\_true and y\_pred.

Example usage:

make\_confusion\_matrix(y\_true=test\_labels, # ground truth test labels

y\_pred=y\_preds, # predicted labels

classes=class\_names, # array of class label names

figsize=(15, 15),

text\_size=10)

"""

# Create the confustion matrix

cm = confusion\_matrix(y\_true, y\_pred)

cm\_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # normalize it

n\_classes = cm.shape[0] # find the number of classes we're dealing with

# Plot the figure and make it pretty

fig, ax = plt.subplots(figsize=figsize)

cax = ax.matshow(cm, cmap=plt.cm.Blues) # colors will represent how 'correct' a class is, darker == better

fig.colorbar(cax)

# Are there a list of classes?

if classes:

labels = classes

else:

labels = np.arange(cm.shape[0])

# Label the axes

ax.set(title="Confusion Matrix",

xlabel="Predicted label",

ylabel="True label",

xticks=np.arange(n\_classes), # create enough axis slots for each class

yticks=np.arange(n\_classes),

xticklabels=labels, # axes will labeled with class names (if they exist) or ints

yticklabels=labels)

# Make x-axis labels appear on bottom

ax.xaxis.set\_label\_position("bottom")

ax.xaxis.tick\_bottom()

# Set the threshold for different colors

threshold = (cm.max() + cm.min()) / 2.

# Plot the text on each cell

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, f"{cm[i, j]} ({cm\_norm[i, j]\*100:.1f}%)",

horizontalalignment="center",

color="white" if cm[i, j] > threshold else "black",

size=text\_size)

Find Probability and make prediction as per labels

# Make predictions with the most recent model

y\_probs = model\_14.predict(test\_data) # "probs" is short for probabilities

# View the first 5 predictions

y\_probs[:5]

y\_probs[0].argmax(), class\_names[y\_probs[0].argmax()]

y\_preds = y\_probs.argmax(axis=1)

# View the first 10 prediction labels

y\_preds[:10]

Now see the confusion Matrix

# Check out the non-prettified confusion matrix

from sklearn.metrics import confusion\_matrix

confusion\_matrix(y\_true=test\_labels,

y\_pred=y\_preds)

make\_confusion\_matrix(y\_true=test\_labels,

y\_pred=y\_preds,

classes=class\_names,

figsize=(15, 15),

text\_size=10)

Plot random image and see model’s prediction

import random

# Create a function for plotting a random image along with its prediction

def plot\_random\_image(model, images, true\_labels, classes):

# Setup random integer

i = random.randint(0, len(images))

# Create predictions and targets

target\_image = images[i]

pred\_probs = model.predict(target\_image.reshape(1, 28, 28)) # have to reshape to get into right size for model

pred\_label = classes[pred\_probs.argmax()]

true\_label = classes[true\_labels[i]]

# Plot the target image

plt.imshow(target\_image, cmap=plt.cm.binary)

# Change the color of the titles depending on if the prediction is right or wrong

if pred\_label == true\_label:

color = "green"

else:

color = "red"

# Add xlabel information (prediction/true label)

plt.xlabel("Pred: {} {:2.0f}% (True: {})".format(pred\_label,

100\*tf.reduce\_max(pred\_probs),

true\_label),

color=color) # set the color to green or red

plot\_random\_image(model=model\_14,

images=test\_data,

true\_labels=test\_labels,

classes=class\_names)

Weights and Biases

model\_14.layers

model\_14.layers[1]

# Get the patterns of a layer in our network

weights, biases = model\_14.layers[1].get\_weights()

# Shape = 1 weight matrix the size of our input data (28x28) per neuron (4)

weights, weights.shape

biases, biases.shape

from tensorflow.keras.utils import plot\_model

# See the inputs and outputs of each layer

plot\_model(model\_14, show\_shapes=True)