```
# SOURCES
# Csv reading: https://www.w3schools.com/python/pandas/pandas_csv.asp

# %%

# --- Imports ------------------------------------------------
import pandas as pd
import numpy as np
from dataclasses import dataclass
# from plotnine import ggplot, aes, geom_tile, geom_point, geom_text, labs, theme_minimal
from plotnine import *

# --- Config -------------------------------------------------
@dataclass
class TrainCfg:
    eta: float = 0.01
    max_epoch: int = 500
    max_epochs_without_improvement: int = 40
    min_delta: float = 1e-5
    base_seed: int = 42

# --- Loading data -------------------------------------------
def load_data(training_csv="training_set.csv", validation_csv="validation_set.csv"):
    training_df = pd.read_csv(training_csv, header=None).to_numpy()
    validation_df = pd.read_csv(validation_csv, header=None).to_numpy()
    x_training = training_df[:,:2].astype(np.float64)
    t_training = training_df[:,2].astype(np.float64)
    x_validation = validation_df[:,:2].astype(np.float64)
    t_validation = validation_df[:,2].astype(np.float64)
    return x_training, t_training, x_validation, t_validation

@dataclass
class Weights:
    W_1: np.ndarray; theta_1: np.ndarray
    W_2: np.ndarray; theta_2: np.ndarray
    w_3: np.ndarray; theta_3: np.ndarray

    def __repr__(self):
        return (f"Weights(\n"
                f"W_1=\n{self.W_1},\n "
                f"theta_1=\n{self.theta_1},\n "
                f"W_2=\n{self.W_2},\n "
                f"theta_2=\n{self.theta_2},\n "
                f"w_3=\n{self.w_3},\n "
                f"theta_3=\n{self.theta_3})")

    def copy(self):
        return Weights(
            self.W_1.copy(),
            self.theta_1.copy(),
            self.W_2.copy(),
            self.theta_2.copy(),
            self.w_3.copy(),
            self.theta_3.copy()
        )

# source: https://en.wikipedia.org/wiki/Weight_initialization
def glorot(fan_in, fan_out, rng):
```

```
    limit = np.sqrt(6.0 / (fan_in + fan_out))
    return rng.uniform(-limit, limit, size=(fan_out, fan_in))

def initializing_weights(M1, M2, rng):
    W_1 = glorot(2, M1, rng)            # shape (M1, 2)
    theta_1 = np.zeros((M1, 1))
    W_2 = glorot(M1, M2, rng)          # shape (M2, M1)
    theta_2 = np.zeros((M2, 1))
    w_3 = glorot(1, M2, rng)            # shape (M2,1)
    theta_3 = np.zeros((1, 1))
    return Weights(W_1,theta_1, W_2, theta_2, w_3, theta_3)

def g(b): return np.tanh(b)
def dgdb(b): return 1-np.tanh(b)**2

def sgn(x):
    if x > 0: return 1
    if x < 0: return -1
    return 0

def forward_pass(x_i, w):
    b_1 =  - w.theta_1 + w.W_1 @ x_i
    V_1 = g(b_1)
    b_2 = -w.theta_2 + w.W_2 @ V_1
    V_2 = g(b_2)
    b_3 = -w.theta_3 + w.w_3.T @ V_2
    O = g(b_3)[0,0]
    return O, V_2, V_1, b_3, b_2, b_1

def update_weights(x_i,t_i, w , eta):
    # Forward pass
    O, V_2, V_1, b_3, b_2, b_1  = forward_pass(x_i, w)

    # Backward pass
    delta_3 = (O-t_i) * dgdb(b_3)
    delta_2 = (w.w_3*delta_3) * dgdb(b_2)
    delta_1 = (w.W_2.T @ delta_2) * dgdb(b_1)

    # Weight updates
    w.w_3 += -eta * delta_3 * V_2
    w.W_2 += -eta * delta_2 * V_1.T
    w.W_1 += -eta * delta_1 * x_i.T

    # Bias updates
    w.theta_3 += eta * delta_3
    w.theta_2 += eta * delta_2
    w.theta_1 += eta * delta_1

    return w

def eval_class_error(w, x_validation, t_validation):
    s = 0
    for x_row, t in zip(x_validation, t_validation):
        x_i = x_row.reshape(2,1)
        t_i = float(t)
        O, *_ = forward_pass(x_i, w)
        s += abs( sgn(O) - t_i )
    return s / (2*x_validation.shape[0])
```

```
def run_perceptron(
        w,
        x_training, t_training,
        x_validation, t_validation,
        cfg,
        rng):
        #eta, max_epoch, max_epochs_without_improvement, min_delta):
    N = x_training.shape[0]
    best_C = np.inf
    best_W = w.copy()
    epoch_since_new_best = 0

    for epoch in range(cfg.max_epoch):
        for i in rng.permutation(N)  :
            x_i = x_training[i].reshape(2,1)
            t_i = t_training[i]
            w = update_weights(x_i,t_i, w, cfg.eta)

        C = eval_class_error(w, x_validation, t_validation)
        # print(f"epoch {epoch:03d} val_err={C:.8f}")

        if C < best_C - cfg.min_delta:
            best_C = C
            best_W = w.copy()
            epoch_since_new_best = 0
        else:
            epoch_since_new_best += 1

        if epoch_since_new_best >= cfg.max_epochs_without_improvement:
            break
    epoch_needed = epoch - epoch_since_new_best
    return best_C, epoch_needed, best_W

def finding_hyper_param(
        M1_grid, M2_grid,
        x_training, t_training,
        x_validation, t_validation,
        cfg, verbose=True):
    rng = np.random.default_rng(seed=cfg.base_seed)
    rows = []
    if verbose:
        print(f"{'M1':>3} {'M2':>3} {'Epoch':>6} {'C':>10}")
    for M1 in M1_grid:
        for M2 in M2_grid:
            w0 = initializing_weights(M1, M2, rng)
            C, epoch, W = run_perceptron(
                w0,
                x_training, t_training,
                x_validation, t_validation,
                cfg,
                rng)
            rows.append({"M1": int(M1), "M2": int(M2), "C": float(C), "epoch": int(epoch)})
            if verbose:
                print(f"{M1:3d} {M2:3d} {epoch:6d} {C:10.4f}")
    return pd.DataFrame(rows)

def plot_M1M2(df, out_png=None):
```

```
    df = df.copy()

    best = df.loc[df["C"].idxmin()]
    best_df = best.to_frame().T

    xbreaks = sorted(df["M2"].unique())
    ybreaks = sorted(df["M1"].unique())
    p = (
        ggplot(df, aes(x="M2", y="M1", fill="C_percent"))
        + geom_tile()
        + geom_text(aes(label="C_lbl"), size=7, color="black")
        # + geom_point(aes(x="M2", y="M1"), data=best_df, size=30, color="black")
        + labs(x="M2", y="M1", fill="C (%)")
        + scale_fill_cmap(cmap_name="viridis_r")
        + scale_x_continuous(trans="log2", breaks=xbreaks, labels=xbreaks, expand=(0, 0))
        + scale_y_continuous(trans="log2", breaks=ybreaks, labels=ybreaks, expand=(0, 0))
        + coord_equal()
        + theme_minimal()
        + theme(
            figure_size=(7, 5),
            axis_text=element_text(size=10),
            axis_title=element_text(size=11),
            panel_grid_minor=element_blank(),
        )
        + guides(fill=guide_colorbar(reverse=True))
    )

    if out_png:
        p.save(out_png, dpi=150)
        print(f"Saved plot to {out_png}")
    return p

def save_csv(df, path):
    df.to_csv(path, index=False)
    print(f"Saved results to {path}")

def load_csv(path):
    return pd.read_csv(path)



# %% LOADING DATA AND INITIALIZING CONFIG ---------------------------------
x_training, t_training, x_validation, t_validation = load_data()
cfg = TrainCfg()


# %% TESTING DIFFERNT M1 & M2 ---------------------------------------------
M1_grid = [1, 2, 4, 8, 16, 32]
M2_grid = [1, 2, 4, 8, 16, 32]
df = finding_hyper_param(M1_grid, M2_grid,
                         x_training, t_training,
                         x_validation, t_validation,
                         cfg)
save_csv(df, "grid_results.csv")


# %% PLOT RESULTS ---------------------------------------------------------
df["C_percent"] = (df["C"]*100).round(3)
```

```
df["C_lbl"] = df["C_percent"].map(lambda x: f"{x:.2f}%")
plot_M1M2(df, "M1M2_comparison.png")


# %% LOAD EXISTING RESULTS ------------------------------------------------
df = load_csv("grid_results.csv")


# %% RUNNING ONE PERCEPTRON -----------------------------------------------
M1 = 4
M2 = 4

rng = np.random.default_rng(seed=40)
W = initializing_weights(M1,M2, rng)
C, epoch, W = run_perceptron(
    W,
    x_training, t_training,
    x_validation, t_validation,
    cfg,
    rng
)



print(f"Epoch: {epoch}")
print(f"C (%): {C*100:.3f}%")
print("Weights:")
print(W)

pd.DataFrame(W.W_1).to_csv("w1.csv", index=False, header=False)
pd.DataFrame(W.W_2).to_csv("w2.csv", index=False, header=False)
pd.DataFrame(W.w_3.reshape(-1,1)).to_csv("w3.csv", index=False, header=False)
pd.DataFrame(W.theta_1.reshape(-1,1)).to_csv("t1.csv", index=False, header=False)
pd.DataFrame(W.theta_2.reshape(-1,1)).to_csv("t2.csv", index=False, header=False)
pd.DataFrame(W.theta_3.reshape(1,1)).to_csv("t3.csv", index=False, header=False)
```