Project 2
Coordinate Descent

Rajasvi Vinayak Sharma
(PID: A59012988)
University of California - San Diego
rvsharma@ucsd.edu

February 21st 2022

1 Problem

Considering standard unconstrained optimization problem $minL(w)$ where $L(\cdot)$ is cost function and $w \in \mathbf{R}^d$, in this project we plan to explore different ways of solving such problems other than standard - gradient descent and stochastic gradient descent - which work under differentiability conditions on $L(w)$. Rephrasing the problem in much simpler way I plan to propose a coordinate descent technique where we initialize a w, followed by repeated process of picking a coordinate $i \in \{1, 2, ..., d\}$ and updating the value of w_i with an aim to reduce the loss. Primarily, we focus on answering two questions: (a) Which coordinate to choose? (b) How to set the new value of w_i?

For this project, Wine Data [Dua and Graff(2017)] will be used to solve binary classification problem for 2 classes $\{0, 1\}$. In the next section, we discuss various techniques to choose & update coordinates (weights).

2 Proposed Approach

In this project we have primarily experimented with 2 approaches and compared the loss with Baseline as Random Weight coordinate descent and Standard Logistic Regression Loss.
Firstly, we propose a weight update strategy where the weight (coordinate) are chosen based on Max Absolute Gradient i.e. choose coordinate direction where change in gradient is maximum. Secondly, we have used a strategy where weights are updated in a round-robin fashion i.e. keep cycling through weights to be updated (choosing a weight and keep updating it for 1 epoch then cycle to next). It is based on intuition that we try to find minimum loss in each coordinate direction one epoch at a time. As far as weight update method is concerned, I have used gradient descent step as the value to update the weights in all cases.

In next sections we will discuss the various approaches in detail along with standard logistic regression and baseline methods (random-feature coordinate descent).

2.1 Standard Logistic Regression

In standard logistic regression technique, we have run stochastic gradient descent based approach without using any regularizer. In this method, at each updation step "all weights" are updated based on calculated gradients. In stochastic approach each updation step takes place at each training example as gradient is re-calculated and weights are updated per training example. Objective is to minimize the following loss function:

$$L(w) = \sum_{i=1}^{n} log(1 + exp(-y^{(i)}(w^T x^{(i)}))) \quad (1)$$

Iterating over training set, at each sample, **all weights** are updated using the following stochastic gradient descent step:

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla g(w_t, x^{(i)}, y^{(i)}) \\ &= w_t + \eta \frac{y^{(i)} x^{(i)}}{1 + exp(y^{(i)}(w^T x^{(i)}))} \end{aligned} \quad (2)$$

2.2 Random-feature coordinate descent

In this method, the coordinate i is chosen uniformly at random then stochastic gradient descent method is used to update the weights w_i at each step. In this strategy the same equation (2) is used but only weight corresponding to randomly chosen coordinates i is being updated.

This random-feature coordinate descent is used as the baseline model with worst-case loss.

2.3 Max Gradient Coodinate Descent

In this first method, we calculate gradient corresponding to each weight and then find the index of weight which has **maximum absolute gradient** value. This is based on intuition that we are trying traverse in the direction/coordinate where the change in gradient (absolute max) is highest. For updating weights, the same equation (2) is being used but only for updating the weight having max gradient with it's actual ∇g value.

$$j = argmax_i(|\nabla g|) \tag{3}$$

$$w_{t+1}^j = w_t^j - \eta max_i(\nabla g(w_t, x^{(i)}, y^{(i)})) \tag{4}$$

where η is learning rate and w^j corresponds to weight which has maximum absolute gradient value. Now, upon each step there is a possibility that new weight is updated which has maximum change in absolute gradient.

Convergence: This method primarily relies on intuition that for each iteration we are trying to choose a weight where we see max change in gradient. Therefore, if we see that for a particular weight/coordinate there is a steepest downfall it makes sense to go in that direction for best result. Similarly, if we see that if this direction is totally opposite (reason behind choosing absolute gradient) then it makes sense to go direction to correct our overall downfall towards optimal minimum loss. Furthermore it can be clearly seen from Fig 1 that loss is finally converging to optimum standard logistic regression loss.

2.4 Round Robin Coordinate Descent

In this second method, we calculate the gradient for coordinate which is already chosen based on iteration. In simple words, we keep updating a single coordinate during an epoch (where in 1 epoch = iterating over all training samples once), and then for next epoch we move onto next coordinate in a cyclic fashion. In other words we are choosing weights in order i.e. $weight_i$ for $i \in \{1, 2, ...13\}, \{1, ..., 13\}$... repeating till all epochs end.

Convergence: This method is based on an intuition that we are trying to find minimum loss in a particular coordinate direction by keeping the coordinate fixed for an epoch and updating the weight corresponding to that direction. In other words, if we try to minimize the loss in each direction individually one at a time then ultimately we should converge to the optimal mimimum loss for overall weights. Therefore keeping the order of iteration constant we keep updating a single weight. Furthermore it can be clearly seen from Fig 1 that loss is finally converging towards standard logistic regression loss.

3 Pseudocode

Algorithm 1. Refers to Max Gradient Coordinate Descent.
Algorithm 2. Refers to Round Robin Coordinate Descent.

4 Experimental results

For all 4 cases i.e. (1) Standard Logistic Regression (2) Random Feature Coordinate Descent (3) Max Gradient Coordinate Descent (4) Round-Robin Coordinate Descent I have run the experiment 100 times to generate sufficient training loss data to evaluate convergence, performance and stability.

Standard Logistic Regression: In this method, stochastic gradient descent is used to update **all weights** while iterating over each training sample. For a single run, 500 epochs (each epoch iterating over all training points) were run resulting in final loss of 0.08. Upon running the experiment for 100 runs, 95% confidence intervals were found to be order of $1 \times e^{-6}$ therefore Mean Loss for Standard Logistic Regression (L^*) is **0.08** $\pm$ **0.0**. Standard Logistic Regression final loss is taken as the best-case loss baseline to compare with our proposed approaches and it can be clearly seen from Fig.1 that rest of the methods

Algorithm 1 Max Gradient Coordinate Descent Method

Input: Training Dataset where Labels ∈ [-1,1]

Output: Minimum Optimal Loss, corresponding weights

```
1: procedure MaxGradientCoordinateDescent($TrainSet, learningRate = \eta$)
2:     $W \leftarrow np.random([NumFeats + 1, 1])$        ▷ Initialize random weights
3:     $Loss \leftarrow L(W)$        ▷ Initialize Loss, eq.(1)
4:     for $e \leftarrow 0$ to $totalEpochs$ do
5:         for $(x, y) \in Dataset$ do
6:             $gradient \leftarrow \nabla g(y, x, W_t)$        ▷ Using eq.(2)
7:             $j \leftarrow argmax(abs(gradient))$        ▷ Select weight index with max absolute gradient
8:             $W_{t+1}^j = W_t^j - \eta * gradeint[j]$        ▷ $j^{th}$ weight update step using eq.(3),(4)
9:             $Loss = L(W_t)$        ▷ Using eq.(1)
10:        end for
11:    end for
12:    return $Loss, W$
13:
```

Algorithm 2 Round Robin Coordinate Descent Method

Input: Training dataset where Labels ∈ [-1,1]

Output: Minimum Optimal Loss, corresponding weights

```
1: procedure RoundRobinCoordinateDescent($TrainSet, learningRate = \eta$)
2:     $W \leftarrow np.random([NumFeats + 1, 1])$        ▷ Initialize random weights
3:     $Loss \leftarrow L(W)$        ▷ Initialize Loss, eq.(1)
4:     $currCoord \leftarrow 0$        ▷ Intiatize current coordinate to update
5:     for $e \leftarrow 0$ to $totalEpochs$ do
6:         for $(x, y) \in Dataset$ do
7:             $gradient \leftarrow \nabla g(y, x, W_t)$        ▷ Using eq.(2)
8:             $j \leftarrow currCoord$        ▷ Select weight index based on Round Robin cycle
9:             $W_{t+1}^j = W_t^j - \eta * gradeint[j]$        ▷ $j^{th}$ weight updated
10:            $Loss = L(W_t)$        ▷ Using eq.(1)
11:        end for
12:        $currCoord \leftarrow (currCoord + 1)\%(numFeats + 1)$        ▷ next coordinate in cycle
13:    end for
14:    return $Loss, W$
15:
```

are converging towards this loss. Furthermore, in the Final Loss vs Number of Runs (Fig.2) it can be seen that final loss for Standard Logistic Regression is consistently lowest (flat line) with all the other methods above it.

Random Feature Coordinate Descent: In this method, random coordinate is chosen at each weight updation step from uniform distribution. Upon running the experiment 100 times with 500 epochs each, it can be clearly seen from Fig.1 that random coordinate descent's loss is decreasing/converging but not exactly becoming equal to standard logistic regression loss. Final Mean loss with 95% confidence interval is found to be **0.74 ± 0.08**. Furthermore, in Final Loss vs Number of Runs graph (Fig.2) it can be seen that this method yield worst final loss in all (100) runs. Hence it can also be seen from loss histogram plot (Fig.3) that random coordinate descent's distribution is on the higher loss end with little overlap with round-robin method.

Max Gradient Coordinate Descent: For this first method, 100 experiment runs with 500 epochs yielded loss which was converging closer to optimal standard logistic regression loss as compared to random baseline worst case. Furthermore, it is clear visible from Loss (log-scale) vs Epochs graph that the this strategy yields 2nd best loss with eminent convergence to L^*. Upon 100 runs, with 500 epochs each, it was concluded that mean loss value for this method is equal to **0.31 ± 0.11** (with 95% CI levels). Furthermore, in the Final Loss vs Number of Runs (Fig.2) it can be seen that final loss for this strategy is consistently 2nd best (2nd lowest) coming closest to standard logistic regression model. Hence, it can also be seen from histogram plot (Fig.3) that it's distribution doesn't overlap with any other method especially it's quite far away from random-feature coordinate descent method.

Round Robin Coordinate Descent: For this second method, 100 experiments runs with 500 epochs each yielded loss which was bar minimum above random feature coordinate descent but still performing better upon final loss. It can be seen from Fig.1 (Loss log-scale vs Epochs) that it's going below random-feature method and seems to be converging with increasing epochs towards minimal loss. Furthermore, it can be seen from Final Loss vs Number of Runs (Fig.2) that it is consistently performing at the least better than random-feature baseline model. Although it can be seen from histogram plot of Final loss (Fig.3) that it's overlapping a little with Random-Feature baseline model but it's final mean loss (with 95% confidence interval) obtained is found to be **0.64 ± 0.05**. Hence, it can be deduced that overall it's better than baseline model.

For each strategy the experiment is run 100 times with 500 epochs each to obtain final loss. The 95% confidence interval is calculated by the following formula:

$$Loss_{lower} = \overline{Loss} - z * \frac{\sigma}{\sqrt{n}} \tag{5}$$

$$Loss_{upper} = \overline{Loss} + z * \frac{\sigma}{\sqrt{n}} \tag{6}$$

$$L(w) = \sum_{i=1}^{n} log(1 + exp(-y^{(i)}(w^T x^{(i)}))) \tag{7}$$

Overall Mean Loss comparison details (along with 95% confidence interval) can be seen in graph Fig.1 and the Table 1. It can be clearly seen that proposed methods i.e. Max Gradient Coordinate Descent and Round-Robin Coordinate Descent perform substantially better than baseline Random-Feature Coordinate Descent Method.

Strategy	Mean Loss (95% CI)
1. Standard Logistic Regression (All Weight)	0.08 ± 0.0
2. Random - Feature Coordinate Descent	0.74 ± 0.08
3. Max Gradient Coordinate Descent	0.31 ± 0.11
4. Round Robin Coordinate Descent	0.64 ± 0.05

Table 1: Comparison Final loss for different proposed methods.

5 Critical Evaluation

From experiments, it can be said that different weight sampling techniques can yield better results as compared to baseline random-feature coordinate descent. For instance, only selecting Max Gradient weight updation we can get close to best case Standard Logistic Regression Model. This approach is a derivation from Gauss-Southwell sampling rule which can be further improved by considering individual weight wise loss. It can be improved by selecting better weight sampling techniques like Lipschitz sampling (selecting weight propotional to loss value for that weight) or by normalising the max gradient method (Gauss-Southwell [Nutini and Schmidt..(2015)]) with loss which is called Gauss-Southwell-Lipschitz sampling.

Furthermore, I believe other methods for weight updation can be used apart from gradient descent update like considering LBFGS or simpler techniques to look for a direction where loss decreases.

In future, I wish to explore other weight sampling and update techniques like Gauss-Southwell-Lipschitz, and LBFGS [Moritz et al.(2016)Moritz, Nishihara, and Jordan] variations.

References

[Dua and Graff(2017)] Dheeru Dua and Casey Graff. 2017. UCI machine learning repository.

[Moritz et al.(2016)Moritz, Nishihara, and Jordan] Philipp Moritz, Robert Nishihara, and Michael Jordan. 2016. A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258. PMLR.

[Nutini and Schmidt..(2015)] Julie Nutini and Schmidt.. 2015. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *International Conference on Machine Learning*, pages 1632–1641. PMLR.

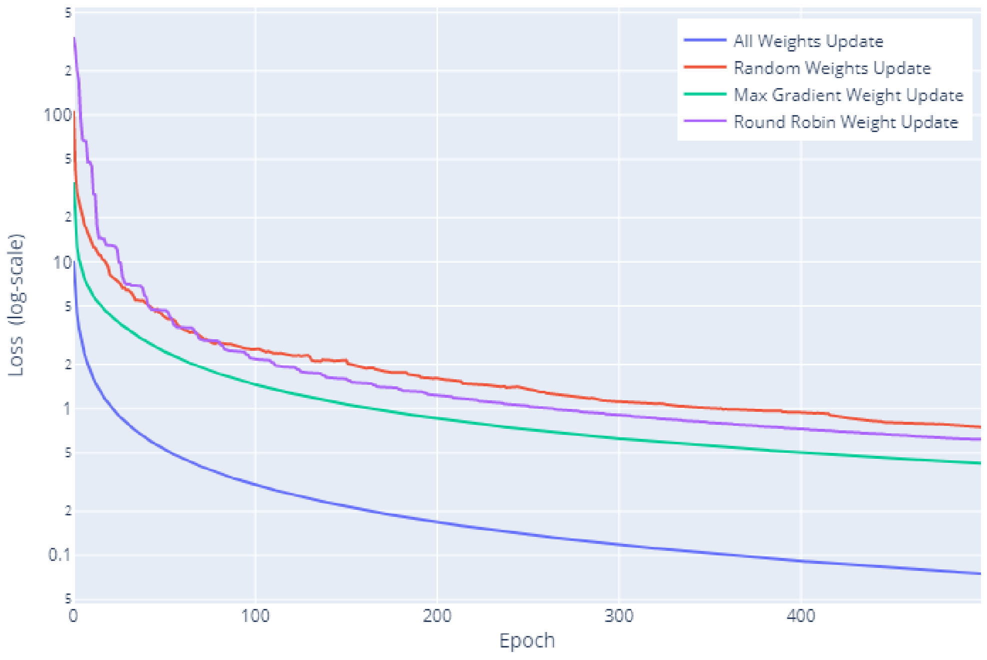

Figure 1: Loss (log-scale) vs Epochs

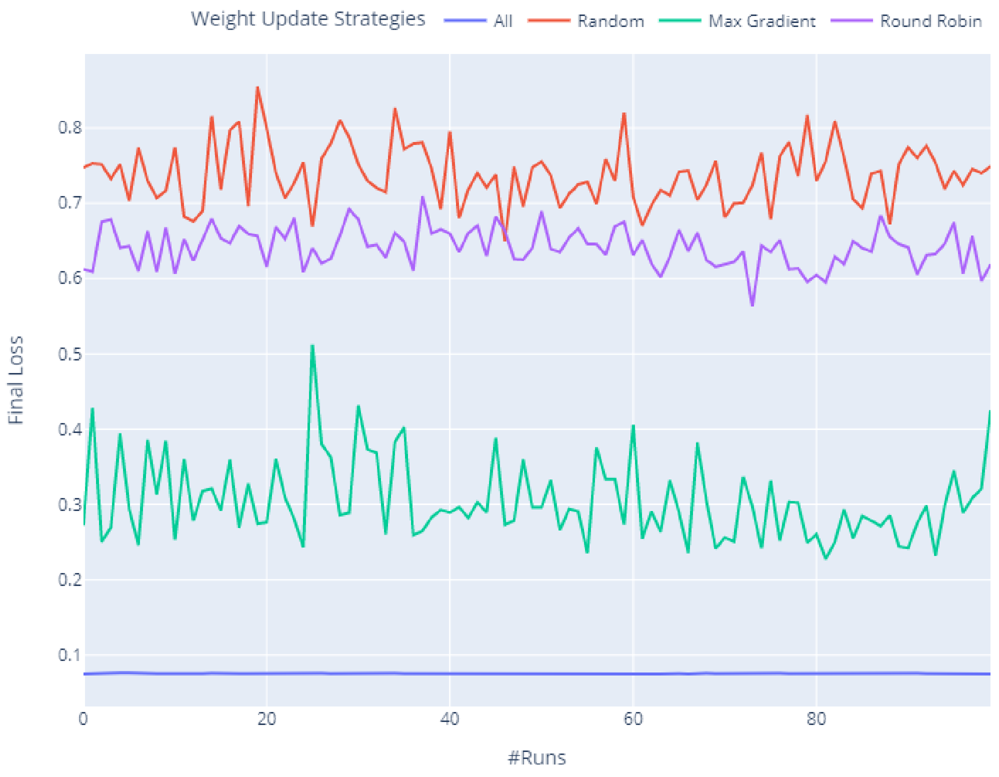

Figure 2: Final Loss vs Number of Runs for each strategy

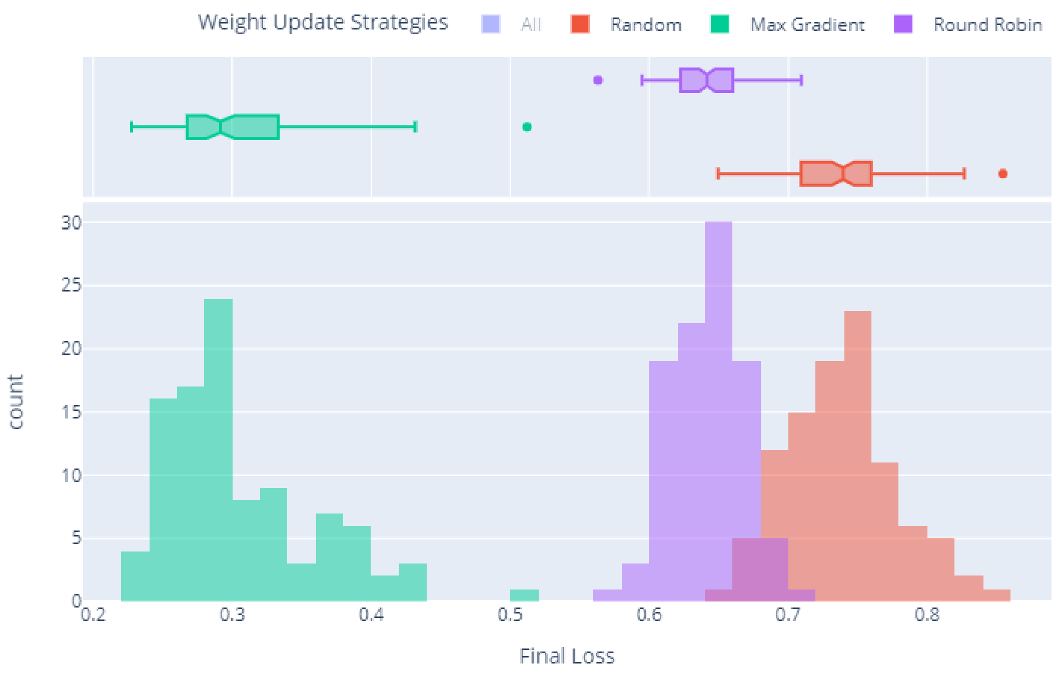

Figure 3: Histogram of Final Loss for 100 runs corresponding to each strategy

Project 2 - Coordinate Descent

February 21, 2022

[1]:
```python
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score,mean_squared_error, log_loss
import plotly.express as px
from sklearn.preprocessing import StandardScaler
import plotly.graph_objects as go
import numpy as np
```

[2]:
```python
data = load_wine(as_frame=True)
X,Y = data.data,data.target
nfeats = len(X.columns)
```

[3]:
```python
# Select only classes 0 and 1
X = X[(Y==0) | (Y==1)].reset_index(drop=True)
Y = Y[(Y==0) | (Y==1)].reset_index(drop=True).values
Y[Y==0]=-1
```

[4]:
```python
def loss_func(x, y, weights):
    return np.sum(np.log(1 + np.exp(-y * (x.dot(weights)))))

def grad_update(
    x, y, weights, feats_selection=[], mode="all", learn_rate=0.1, curr_coord=0
):
    assert weights.shape == (nfeats + 1, 1)

    if x.shape != (nfeats + 1, 1) and mode != "all":
        x = x.reshape([-1, 1])

    if mode == "all":
        gradient = -np.sum((y * x) / (1 + np.exp(y * (x.dot(weights)))), 0).
 ↪reshape(
            [-1, 1]
        )
    else:
        gradient = -(y * x) / (1 + np.exp(y * (weights.T.dot(x)))).reshape([-1,␣
↪1])

    if feats_selection == "random":
        temp = np.zeros(gradient.shape)
        random_coord = np.random.choice(14)
        temp[random_coord] = gradient[random_coord]
        gradient = temp

    elif feats_selection == "max":
        t = np.argmax(abs(gradient))
        gradient[gradient != gradient[t]] = 0

    elif feats_selection == "round_robin":
        temp = np.zeros(gradient.shape)
        temp[curr_coord] = gradient[curr_coord]
        gradient = temp

    return weights - (learn_rate * gradient.reshape([-1, 1]))
```

[ ]:
```
runs = 100
nfeats = len(X.columns)
epochs = 500
fin_loss = []

for i in range(runs):
    curr_coord = 1
    fig = go.Figure()

    X_train, y_train = X.copy(), Y.copy()

    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = scaler.transform(X_train)

    X_train = np.append(X_train, np.ones([len(X_train), 1]), 1)
    y_train = y_train.reshape([-1, 1])

    weights = np.random.random([nfeats + 1, 1])
    weightsRandomUpdate = weights.copy()
    weightsMaxGradUpdate = weights.copy()
    weightsRR = weights.copy()

    loss = []
    lossMaxGradUpdate = []
    lossRandomUpdate = []
    lossRR = []
    for e in range(epochs):
        for it in range(len(X_train)):

            # All weights update
            weights = grad_update(X_train[it], y_train[it], weights,␣
↪mode="stochastic")
            curr_loss = loss_func(X_train, y_train, weights)

            # Random weights update
            weightsRandomUpdate = grad_update(
                X_train[it],
                y_train[it],
                weightsRandomUpdate,
                feats_selection="random",
                mode="stochastic",
            )
            curr_loss_rand_update = loss_func(X_train, y_train,␣
↪weightsRandomUpdate)

            # Only Maximum Gradient Weight Update
            weightsMaxGradUpdate = grad_update(
                X_train[it],
                y_train[it],
                weightsMaxGradUpdate,
                feats_selection="max",
                mode="stochastic",
            )
            curr_loss_max_grad = loss_func(X_train, y_train,␣
↪weightsMaxGradUpdate)

            # Round Robin Weight Update
            weightsRR = grad_update(
                X_train[it],
                y_train[it],
                weightsRR,
                mode="stochastic",
                feats_selection="round_robin",
                curr_coord=curr_coord,
            )
            curr_loss_rr = loss_func(X_train, y_train, weightsRR)

        curr_coord = (curr_coord + 1) % (nfeats + 1)

        loss.append(curr_loss)
        lossRandomUpdate.append(curr_loss_rand_update)
        lossMaxGradUpdate.append(curr_loss_max_grad)
        lossRR.append(curr_loss_rr)

        if e % 499 == 0 and i==0:
            print(
                f"Epoch {e} :: Current Loss: {curr_loss:0.2f} ::␣
↪{curr_loss_rand_update:0.2f} :: {curr_loss_max_grad:0.2f} :: {curr_loss_rr:0.
↪2f}"
            )

    fin_loss.append(
        (curr_loss, curr_loss_rand_update, curr_loss_max_grad, curr_loss_rr)
    )

    fig.add_trace(
        go.Scatter(
            x=list(range(epochs)), y=loss, mode="lines", name=f"All Weights␣
↪Update"
        )
    )

    fig.add_trace(
        go.Scatter(
            x=list(range(epochs)),
            y=lossRandomUpdate,
            mode="lines",
            name=f"Random Weights Update",
        )
    )

    fig.add_trace(
        go.Scatter(
            x=list(range(epochs)),
            y=lossMaxGradUpdate,
            mode="lines",
            name=f"Max Gradient Weight Update",
        )
    )

    fig.add_trace(
        go.Scatter(
            x=list(range(epochs)),
            y=lossRR,
            mode="lines",
            name=f"Round Robin Weight Update",
        )
    )
```

```
fig.update_layout(
    autosize=False,
    width=800,
    height=600,
    legend=dict(yanchor="top", y=0.98, xanchor="right", x=0.99),
)

fig.update_xaxes(title_text="Epoch", title_standoff=5)
fig.update_yaxes(title_text="Loss  (log-scale)", title_standoff=5,type="log")
fig.show()
```

```
Epoch 0 :: Current Loss: 10.77 :: 89.38 :: 33.05 :: 253.11
Epoch 499 :: Current Loss: 0.08 :: 0.75 :: 0.35 :: 0.67
```

[ ]:
```
# Final Loss Graphs
leg = ["All","Random","Max Gradient","Round Robin"]

df=pd.DataFrame(fin_loss)
df.columns=leg
df=df.reset_index()
df=pd.melt(df,id_vars=["index"],value_vars=leg)
df=df.rename(columns={"value":"Final Loss","index":"#Runs","variable":"Weight␣
 ↪Update Strategies"})

# Final Loss vs No. of Runs Plot
fig=px.line(df,x="#Runs",y="Final Loss",color="Weight Update␣
 ↪Strategies",height=600,width=800)
fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))
fig.show()
```

[ ]:
```
# Final Loss Distribution Plot
fig = px.histogram(df,x="Final Loss",color="Weight Update␣
 ↪Strategies",height=500,width=800,nbins=50,barmode="overlay",marginal="box")
fig.update_layout(legend=dict(
    orientation="h",
    yanchor="bottom",
    y=1.02,
    xanchor="right",
    x=1
))
fig.show()
```

[ ]:
```
print("Mean Final Loss (with 95% confidence interval): ")
for i,l in enumerate(leg):
    print(f"{l} Weight Update Strategy: \n {np.mean(fin_loss,0)[i]:0.2f} +-␣
 ↪{2*np.std(fin_loss,0)[i]:0.2f}    ")
```