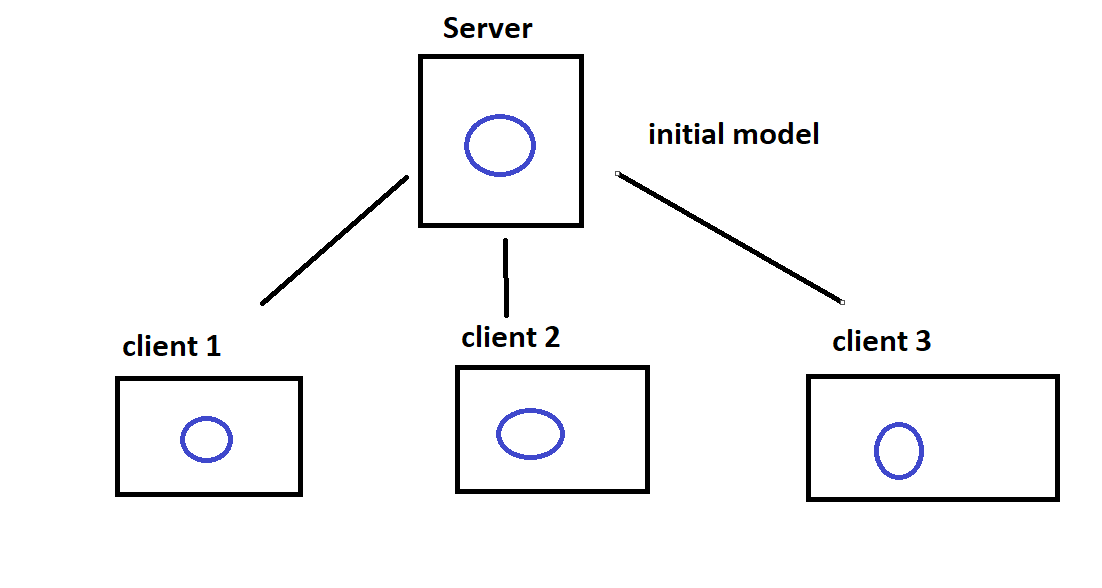
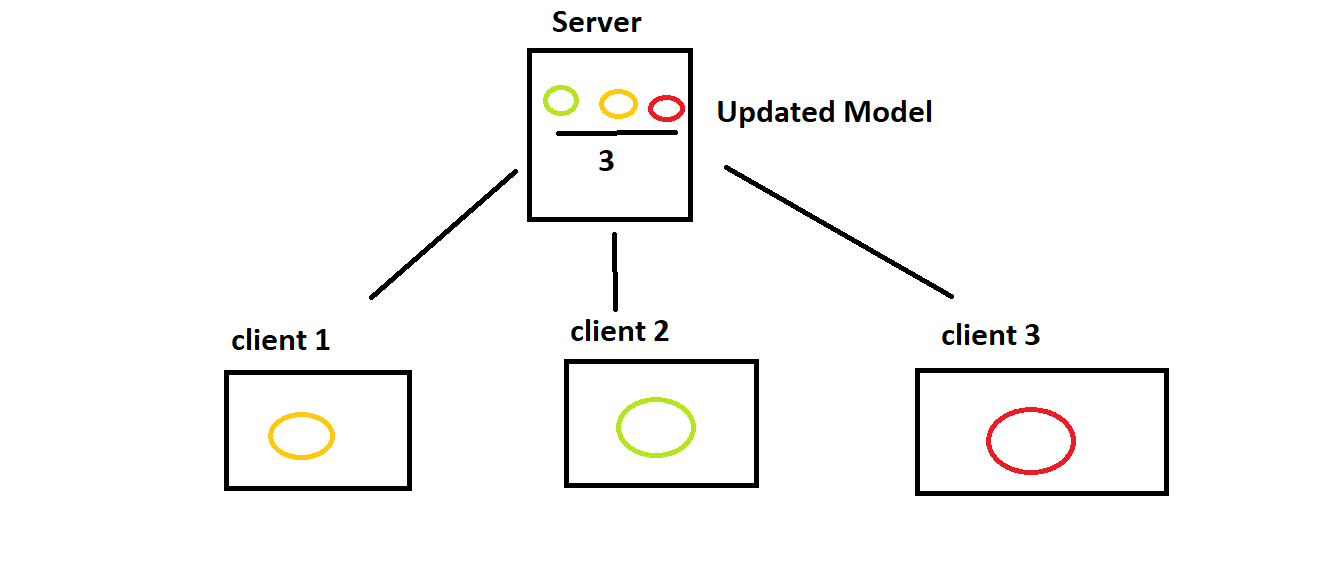
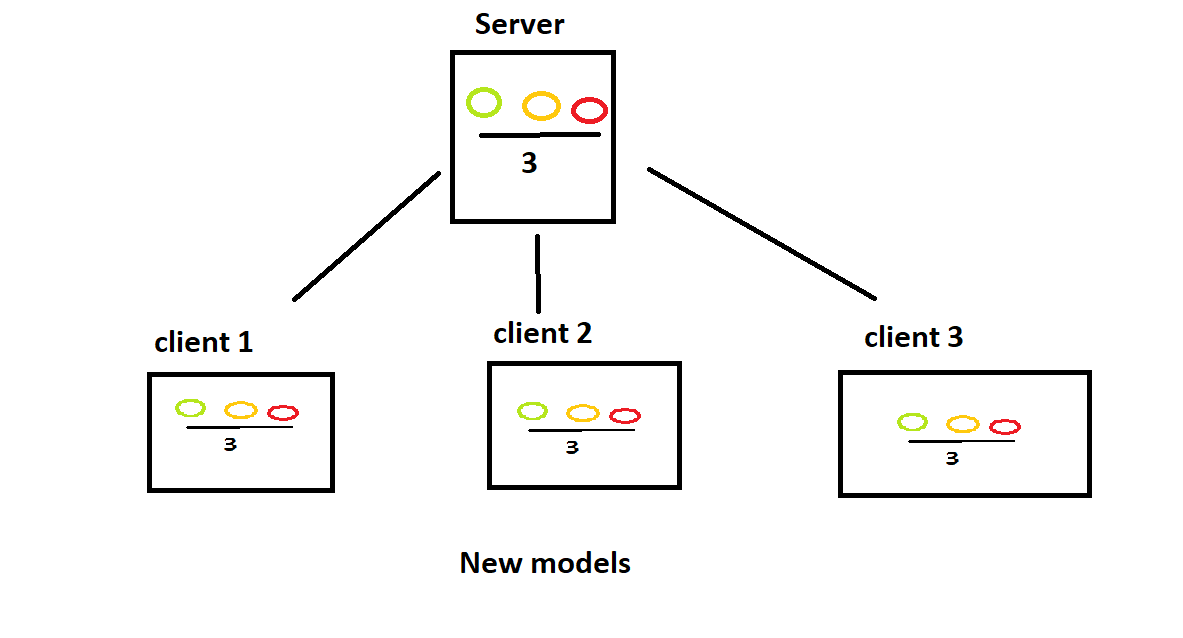
**Federated Learning**

**Basic Idea:**

1. We have a global server and lots of clients.
2. Generally in centralized ML, clients send data to server, server is trained on the basis of the data received and the output is sent back to client.
3. This can cause breach of privacy and requires the server and client to remain connected most of the time.
4. Introducing Federated Learning: Here we train the model locally on each client and send the model updates to the server, where server just averages the values received and sends the updated model back to the clients. Hence all the processing (aka learning) stuff can be done locally.
5. This is important if we look from the perspective of privacy and the interdependence between server and client reduces vastly.

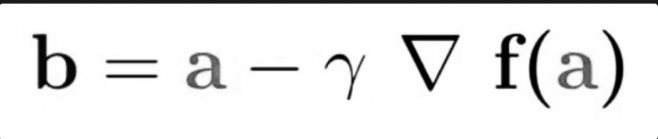
** **

**Step 1 Step 2**

****

**Step 3**

**Before moving ahead with the required algorithms, lets understand GRADIENT DESCENT.**

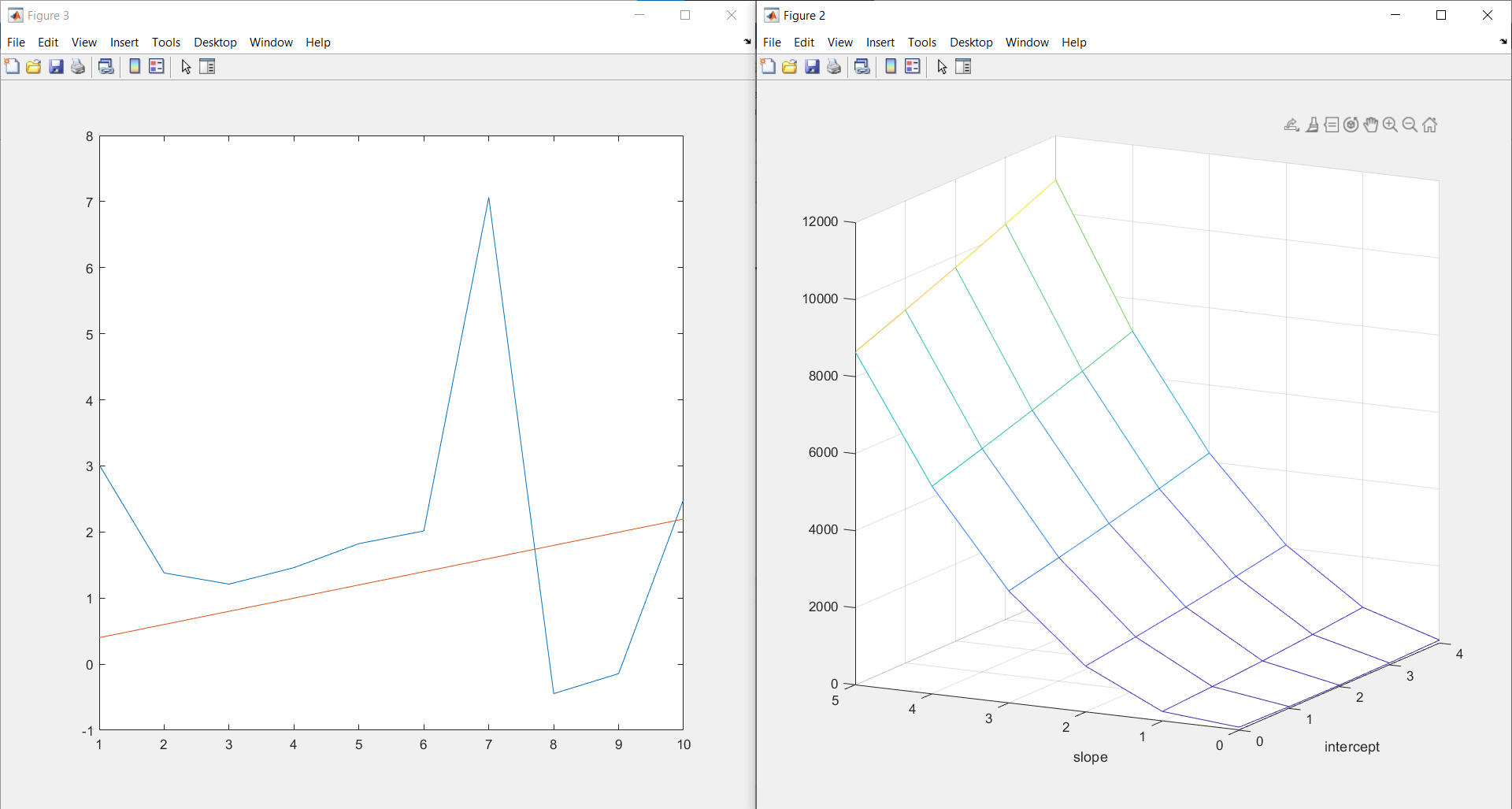
**GRADIENT DESCENT:**

Okay what is this??

Imagine you’ve been given a cost function (or loss function if you may). We want to minimize our losses in a model, right? Now the region of minimum error would be the one where the loss function has the **LEAST** gradient aka slope.

Now in the above equation, **b** is a point on the function where we want to move next, so as to reduce the slope from the previous point **a**. **Δf(a)** is the **direction of the steepest descent** at point **a. Gamma** is a waiting factor that determines the learning rate, which is important as too fast of a learning rate may result in pretty bad optimizations and too slow of a learning rate would be, well, slow.

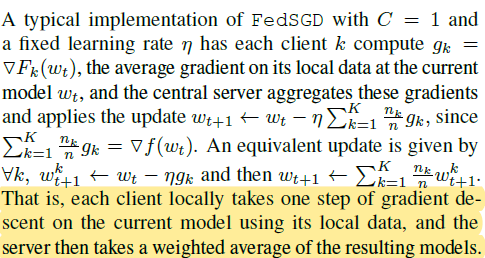
<https://builtin.com/data-science/gradient-descent>



**Fed SGD (Stochastic Gradient Descent)**

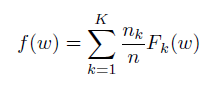
Okay this is like too many big words in a sentence right now. Let’s take it in slowly. First things first, this process is straight up **naïve**, and not very useful. However, for the sake of understanding:

1. Imagine 3 clients and one server.
2. We’ll work it out for a **C=1** (C being fraction of clients used).
3. Imagine each client returns a gradient descent g1,g2,g3 respectively,
4. Now comes the fun part. Remember the loss function? Assume it to be **f.**
5. **fnew = fold – η\*(g1+g2+g3)/3 (η is learning rate)** (repeat this back and forth)
6. This process happens on the server side, and it is not hard to imagine that this process requires **FREQUENT** communication between server and client (talk about inefficiency).
7. For C=2/3, take any two clients and repeat.



Something fancy McMahan wrote, but means essentially the same.

**FedAVG**

****

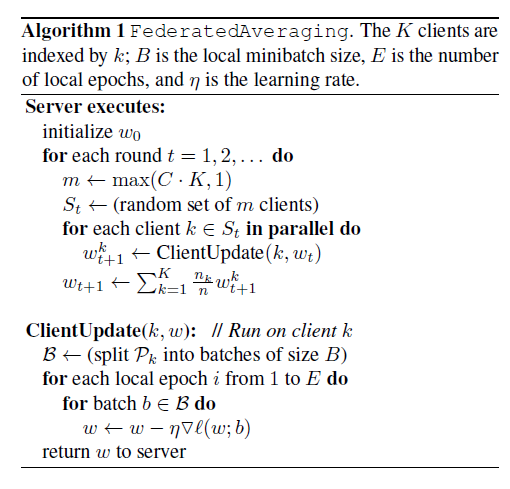
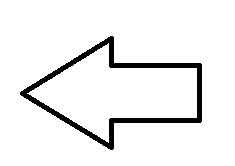
Global loss function needed to be minimized

K- number of clients

Fk- Loss function of each client

nk- weight of each client’s loss (larger datasets have more losses)

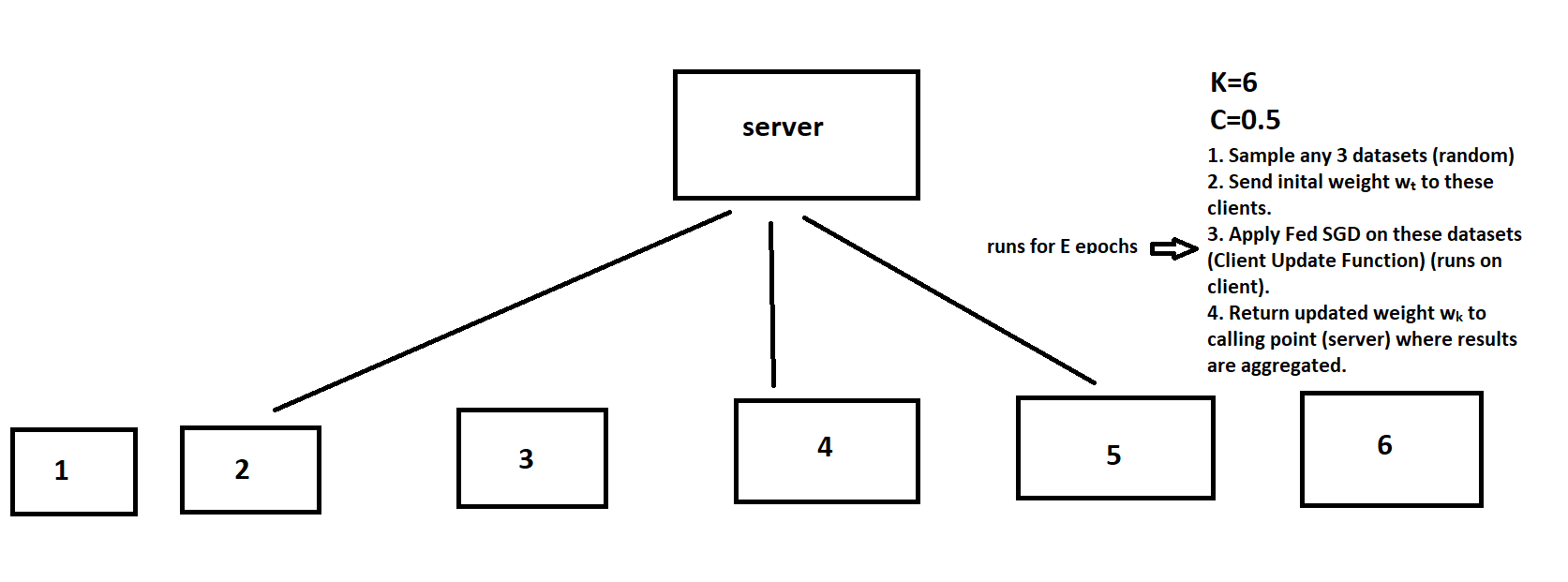
w- weight (how much influence a given input will have on the output)



Okay broadly speaking, FedAVG is basically Fed SGD, but with the concept of **mini batches** and **epochs**. **B** decides the minibatch size and **E** is the number of epochs (number of iterations that the algorithm will run through the dataset).

Let’s break it down:

Assume C=0.5 and K=6 (C.K=3). Round 1:



Round 2: Follow similar steps for next 3 samples. Repeat for more accuracy.

We can see that at a particular instance, not all clients are required to send their calculated Fed SGD weights to the server. Hence this ensures that client and server are NOT interdependent ALL THE TIME. Also in real life, data is never IID (independent and identically distributed), so using Fed AVG makes sense as setting small batch sizes helps in increasing parallelism (as multiple epochs can be run before the actual averaging takes place).