Stochastic Gradient Descent:

Stochastic gradient descent is an alternative to classic (or batch) gradient descent and is more efficient and scalable to large data set.

Algorithm:

1. Randomly shuffle the dataset.

2. For $i=1, \ldots, m$

$$\Theta_j := \Theta_j - \alpha \left(h_\Theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}$$

Stochastic gradient descent will be unlikely to converge at the global minimum and will instead wander around it randomly, but usually yields a result that is close enough.

Mini-Batch gradient descent can sometimes be even faster than stochastic descent.

This time we'll use $b = 2 - 100$, lets $b = 10$

Algo: Repeat {

For $i = 1, 11, 21, \ldots m$

$$\Theta_j := \Theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h_\Theta(x^{(k)}) - y^{(k)} \right) x_j^{(k)}$$

}

With a smaller learning rate α, it is possible that we may get a slightly better solution with stochastic gradient descent.

One effective strategy for trying to actually converge at the global minimum is to slowly decrease α over time. For example

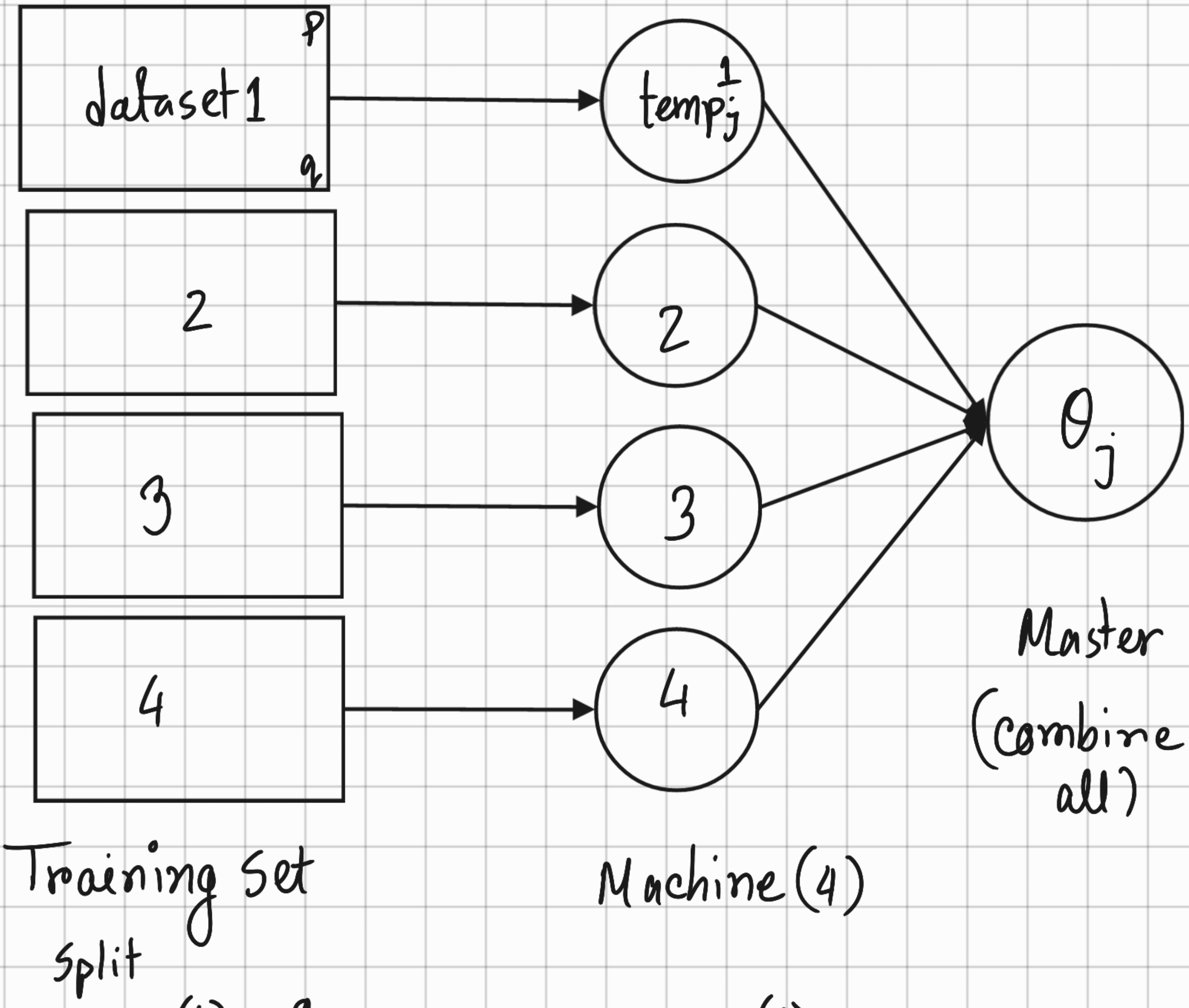
$$\alpha = \frac{\text{cons1}}{\text{iterationNumber} + \text{cons2}}$$

Online learning:

Instead of train whole dataset at a time. we train one by one. In a website, using CTR(Click through Rate) we can update our model after one click of user based on preference. Example: In a search query let's say 10 results shown, & user click on a particular result, then we can update our model based on the search query (X) & the particular result ($y=1$)

Map reduced and data parallelism

We can split our whole training set into z subset based on how many machine we have.



$$temp_j^{(i)} = \sum_{i=p}^{q} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} \qquad i = 1, 2, 3, 4$$

$$\theta_j := \theta_j - \alpha \frac{1}{4} \left(temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)}\right)$$

For all $j = 0, \ldots n$