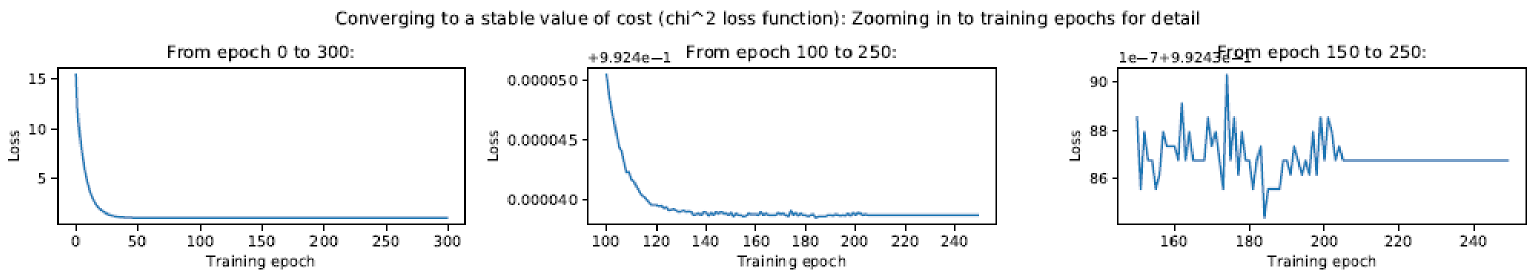
Linear Regression Assignment

1. **How many training cycles are required for the fit to converge to a stable value of the cost?** (This is for data with target gradient **m=2.0**, target intercept **c=0.5**, and no. of examples (Ngen)=100)

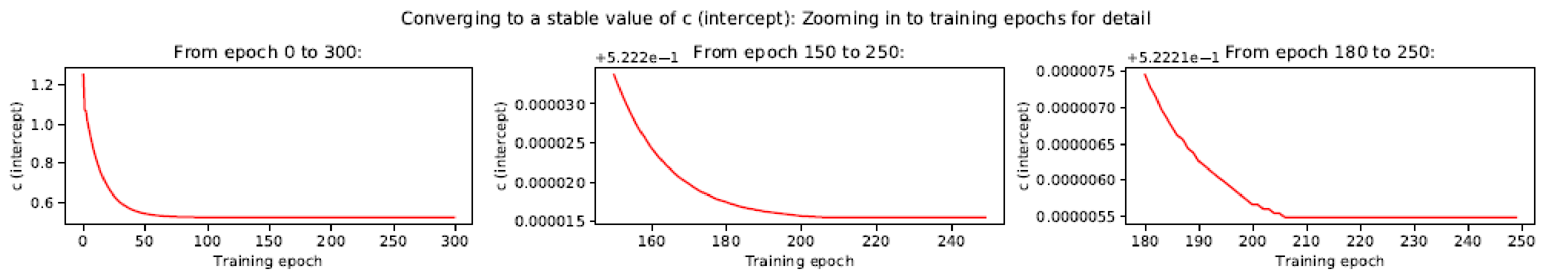
By looking at the verbose print output (uncommenting line 99 in Example_LinearRegression.py), we can inspect each of the outputs for the cost (loss function), gradient and intercept, over each training cycle. When the number of training cycles is set to 300 we see that by the end of the training session, we converge to a fit where the cost is 0.9924 to 4 decimal places. The optimiser has been hovering around this value (±0.01) since cycle around the 50th training epoch. We can see from the left-hand plot below that sometime after the 50th epoch, the value of the loss changes only a little.

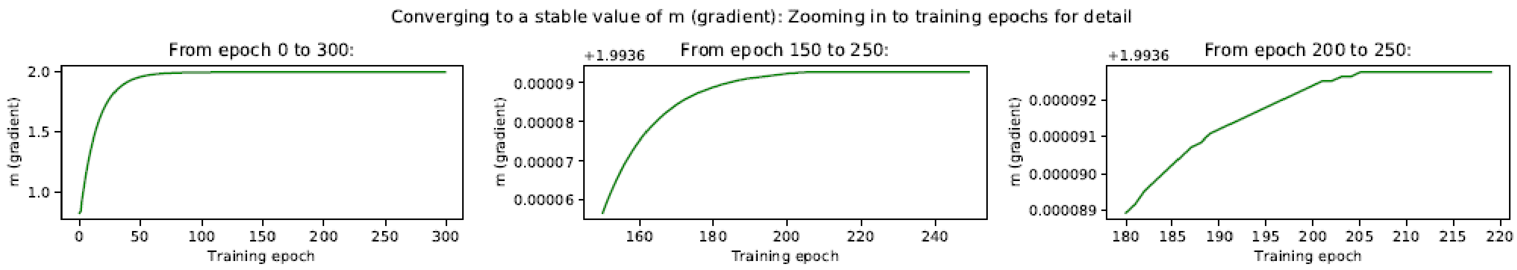
However, the console readout allows us to see values up to 7 decimal places (loss = 0.9924387 in this instance). When we zoom in to the curve of the graph (centre plot and right-hand plot below), we can see the changes that are being made on each further training cycle as these extra few decimal places are optimised; the less continues to decrease even after 100 training cycles. The loss does not completely plateau until just after the 200th training epoch.



Therefore, depending on the level of precision you wish to obtain, you may wish to run only 100 cycles for a reasonable answer, or 250 cycles to reach the limit of this function's precision. Note, however, that this tiny (effectively negligible) increase in performance may be consistent with overfitting, and it would be beneficial to cross-validate the model against a set of test data.

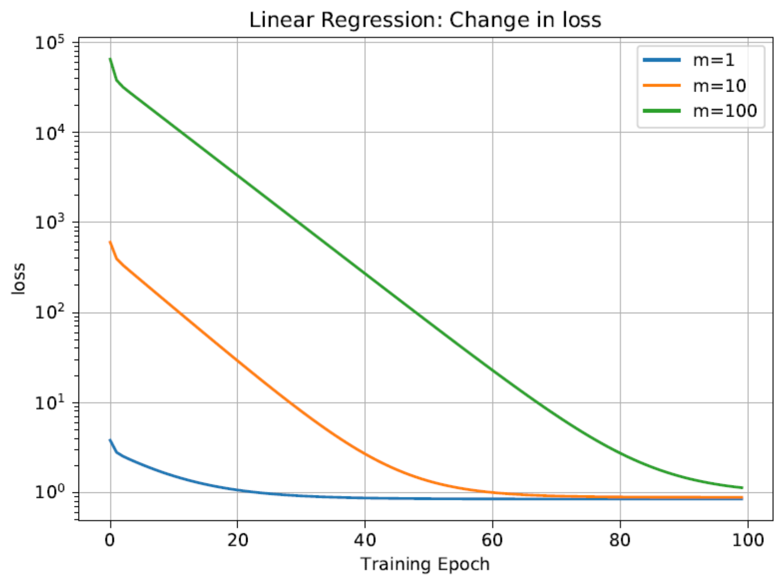
Similar observations can be made for the convergence behaviour of the fit for the intercept, c (optimal value = [0.9924387]), and the gradient, m (optimal value = [0.5222155]).





1. **Investigate the fit convergence behaviour for the following scenarios, where the intercept and gradient {c, m} are {0, 1}, {0, 10}, and {0, 100}:**

The convergence behaviour for these is similar to that of the above example (c=0.5, m-2) with respect to the speed at which they converge (i.e. number of training epochs required to reach optimal values is <100).



The following difference in behaviour between the three cases is observed: Although the number of cycles taken to converge is similar in all cases, the scale of the change between epochs becomes much greater as the target gradient is increased. When m=1, the loss produced at the end of the first epoch is in the order of 10^1. But when m=10, the loss after one epoch is in the order of 10^3, and when m=100 it is around 10^5. As the target gradient increases, the initial cycle produces much greater loss.

This is because as the target gradient increases, our initial value for the parameter m is now further and further away from ground truth. The initial parameters c and m are set by a single random number. The random number generated is usually around the region of 0-1. When these initial values go through the linear regression algorithm during the first training epoch, they will be adjusted to some degree, but if they start off close to the ground truth, then they will not need to go through such an extreme amount of change before the function converges on an optimal solution.

Therefore, when m=1, and the initial parameter for m is around 1, the loss after the first epoch will not be very great (around 10^1). However, when the ground truth is m=100 but the initial value of m is around 1, the loss after the first epoch will be very large (around 10^5).

Since m and c are correlated in y=mx+c, if the value of m after the first epoch is a value that is very far from ground truth, then c will need to be pushed in the opposite direction in order to balance out the loss. This is why we observe the parameter c being set to very high values in the first few epochs, to offset the inadequately low m. This then evens off after 50+ epochs (But why is it never the other way around (very high m, very low c)?

[Note: the reading on these graphs starts from AFTER the first epoch, so the first epoch value for m is NOT equal to the randomly generated number, but equal to the first adjustment performed on this by the linear regression algorithm.]