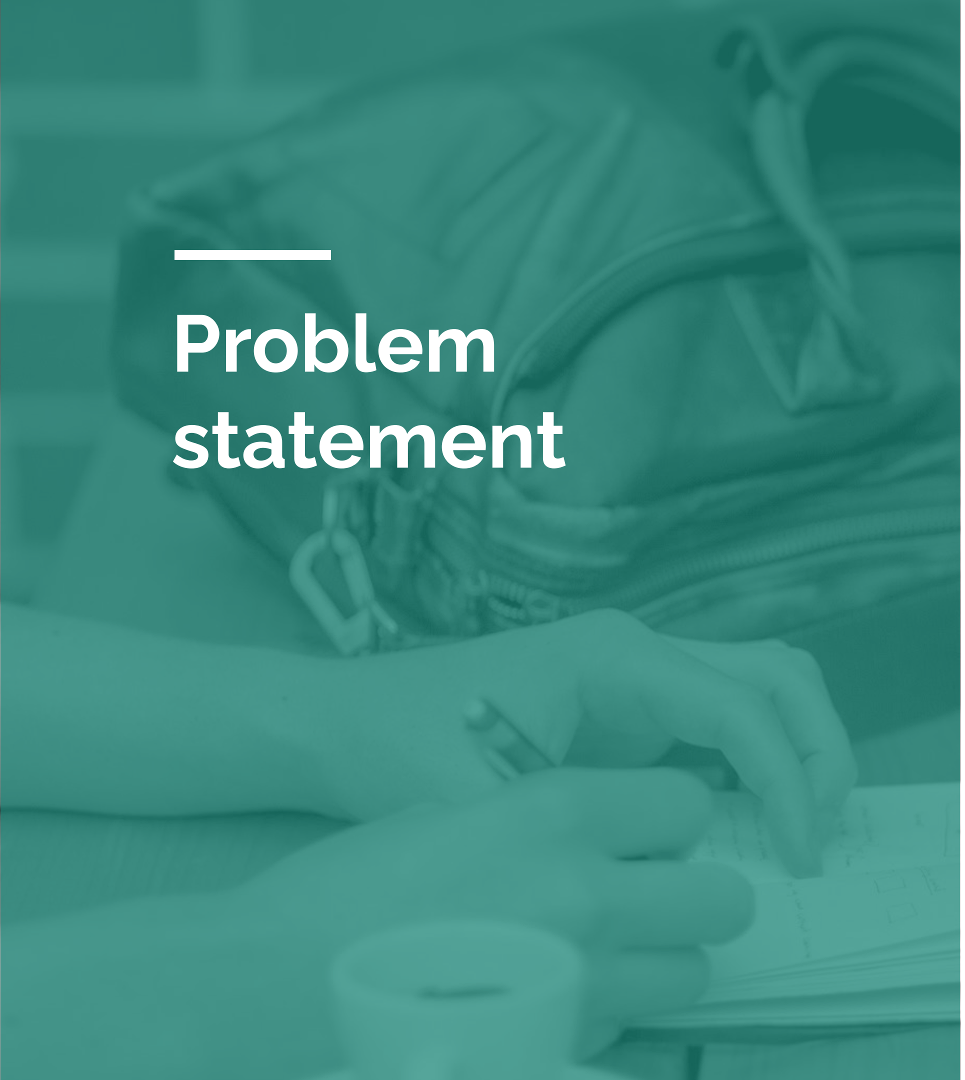
Binary Classification

Using Gradient-based optimization algorithms on binary Cross-Entropy loss function



Problem statement

Our main goal is to see if a customer subscribes to a term deposit or not in a bank.

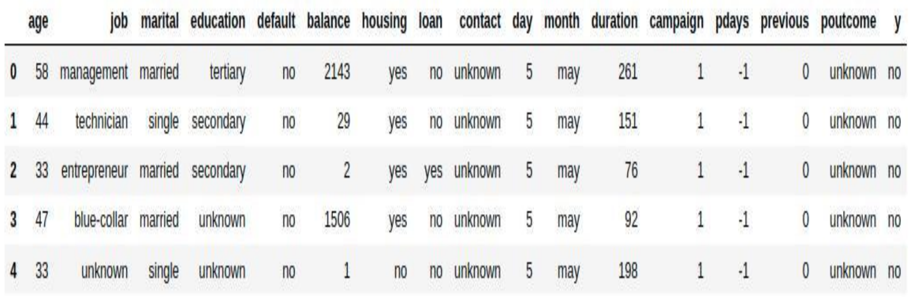
We are going to model the probability of a person subscribing to the term deposit by several explanatory variables like age, job type, personal and home loan status etc. which are both numerical and categorical in nature.

DATA

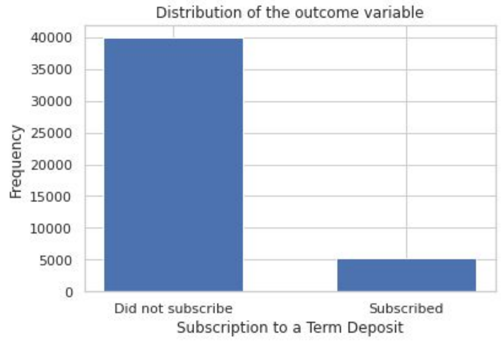
Source and snippet

SOURCE

https://www.kaggle.com/datasets/sonujha090/bank-marketing?select=bank-full.csv



	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no



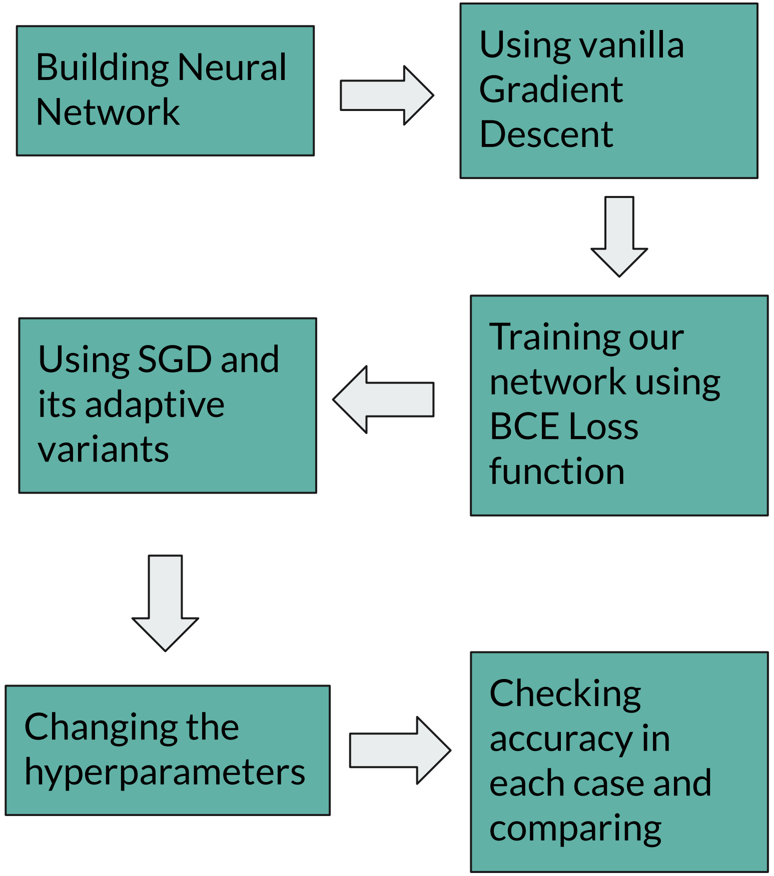
MODEL AND LOSS FUNCTION

We have 16 features and 45211 instances in our data. In the preprocessing phase we have discarded two features. We have chosen a simple Neural network (to avoid overfitting) consisting 4 Linear layers and ReLU as our activation function, since the data dimension is not huge.

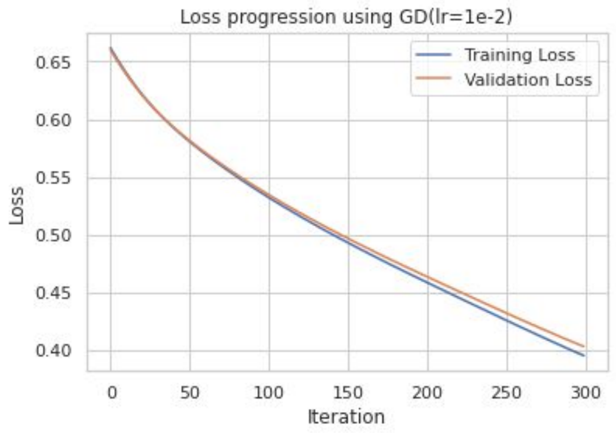
We have chosen binary cross-entropy loss to go with. The formula of the loss function is given by,

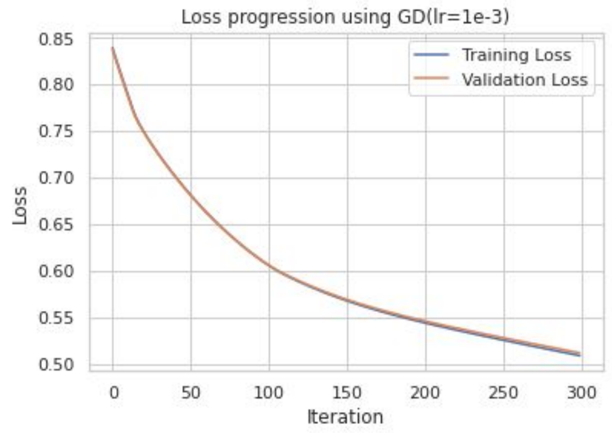
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

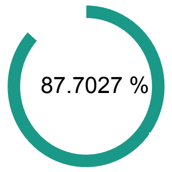
WORKFLOW

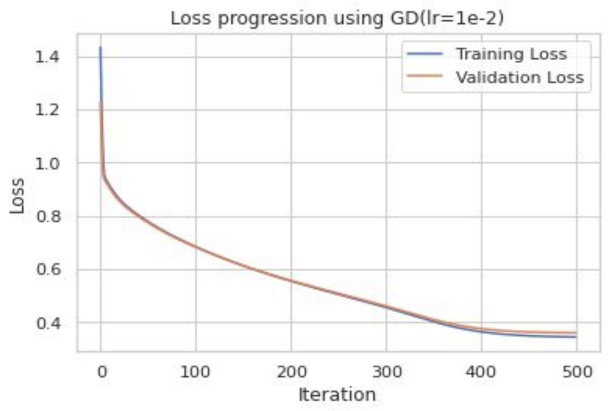

Building Neural Network
Using vanilla Gradient Descent
Training our network using BCE Loss function
Using SGD and its adaptive variants
Changing the hyperparameters
Checking accuracy in each case and comparing

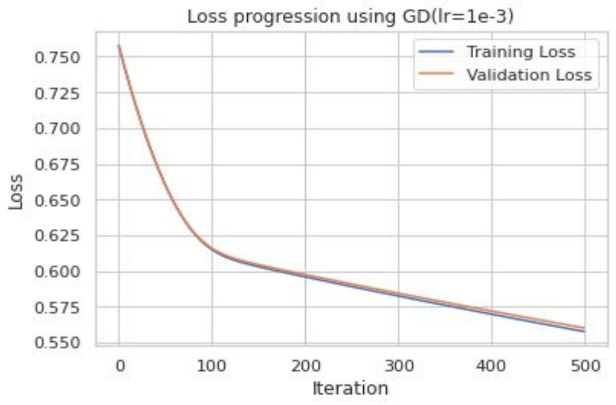
GRADIENT DESCENT





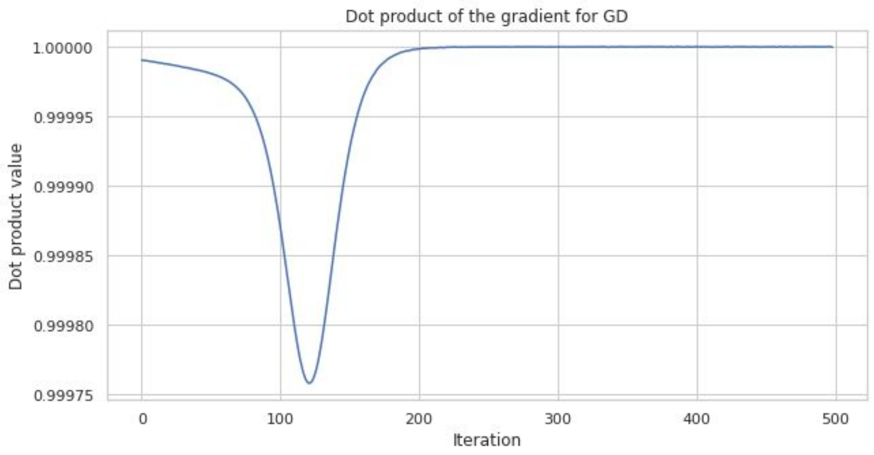


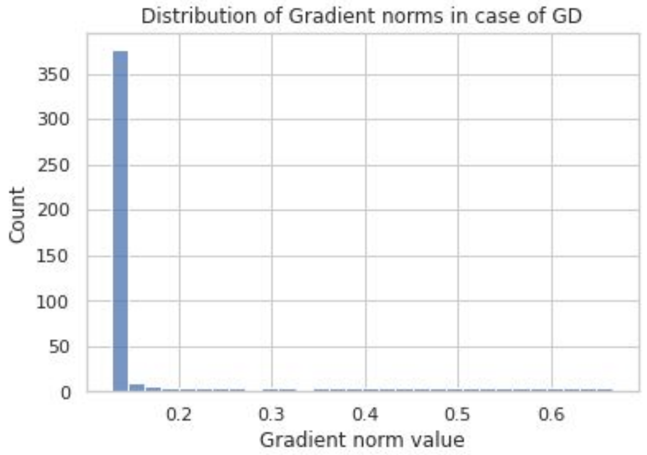




In Gradient descent we work with the whole batch of training sample. We propagate towards negative gradient calculated from the whole. When we used learning rate 1*10^-2 and 500 epochs, it seems the algorithm has reached the minima and achieves maximum accuracy, as opposed to the other cases it still shows a diminishing trend.

GRADIENT DESCENT (continued…)

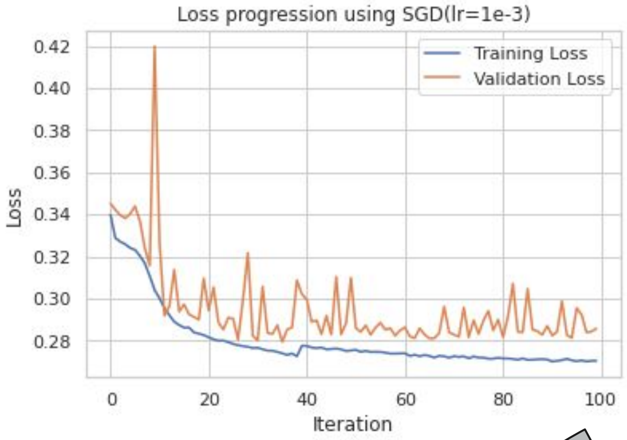


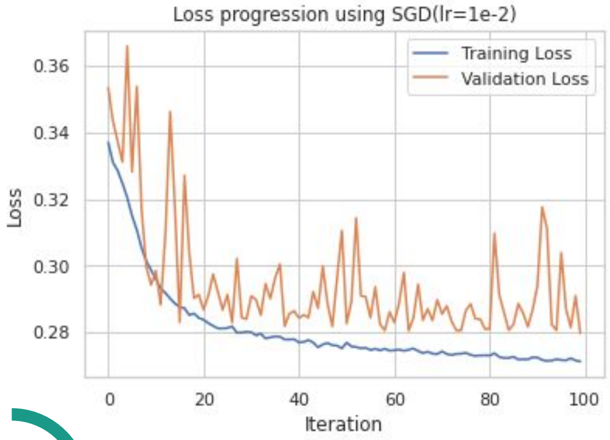


Above, we show how the direction of the gradient corresponding to the last layer of the neural network changes with epochs. As expected the cosine value of the angle between gradients is very close to one or equal to one, that is the algorithm chooses more or less similar path in each epoch in the search space to find the minima.

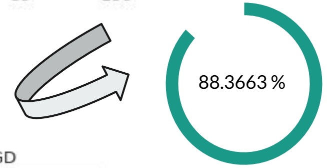
The above histogram shows the distribution of the norm of the gradients in the last linear layer. It shows maximum density at zero and the curve diminishes or vanishes after zero.

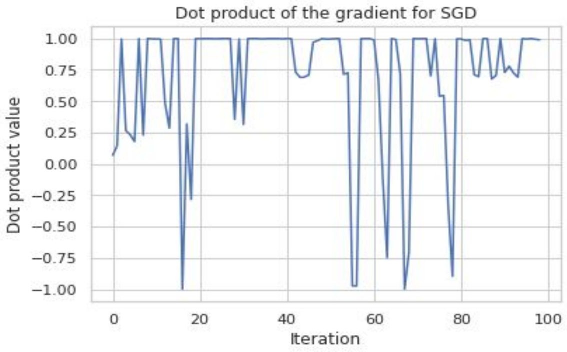
STOCHASTIC GRADIENT DESCENT

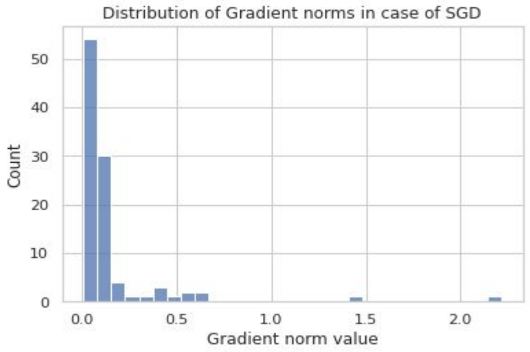




In SGD the gradient is calculated by selecting an observation from the training set randomly, so the the approximated gradient may be different from the actual gradient calculated using the whole training set. That is why the loss progression fluctuates continuously.

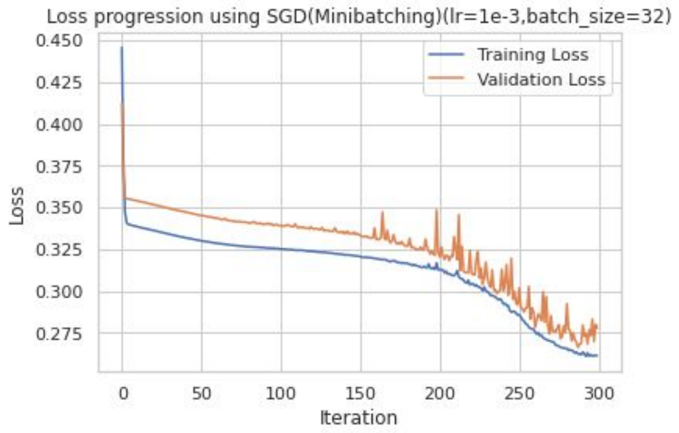


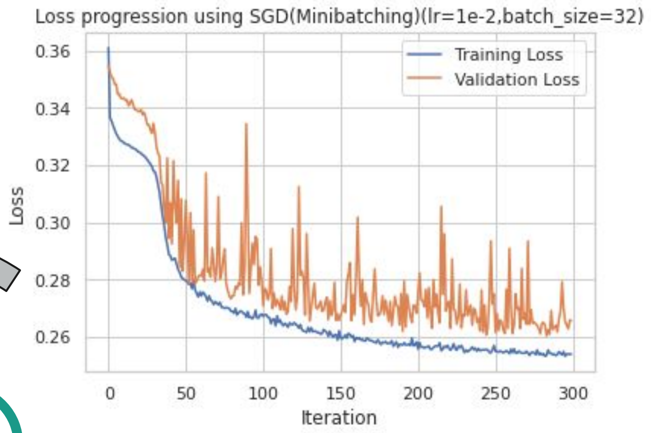


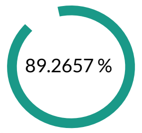


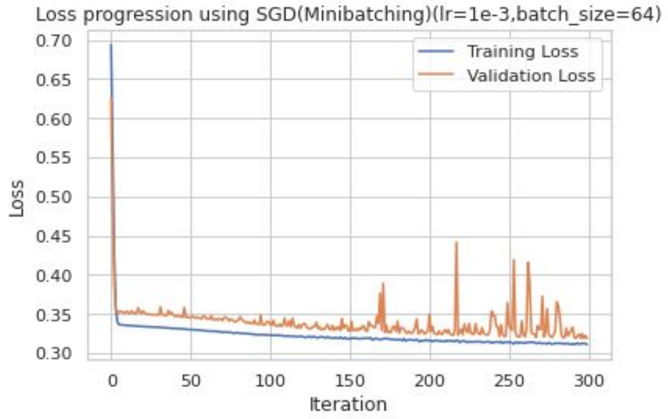
As we can see for SGD the gradient direction in each epoch is not same as the previous and changing direction completely sometimes, also from the histogram it is clear that the estimated gradient is not unbiased estimator of the actual gradient.

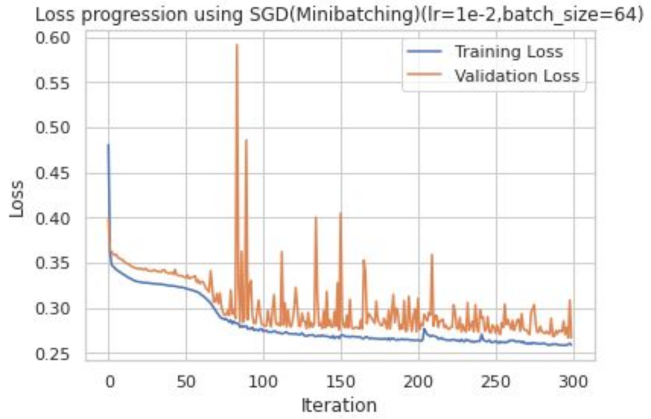
SGD WITH MINI-BATCHING





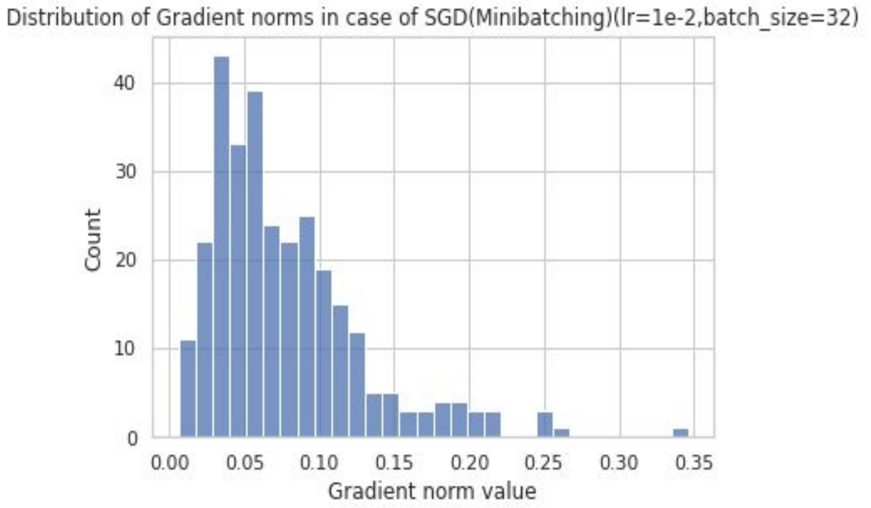


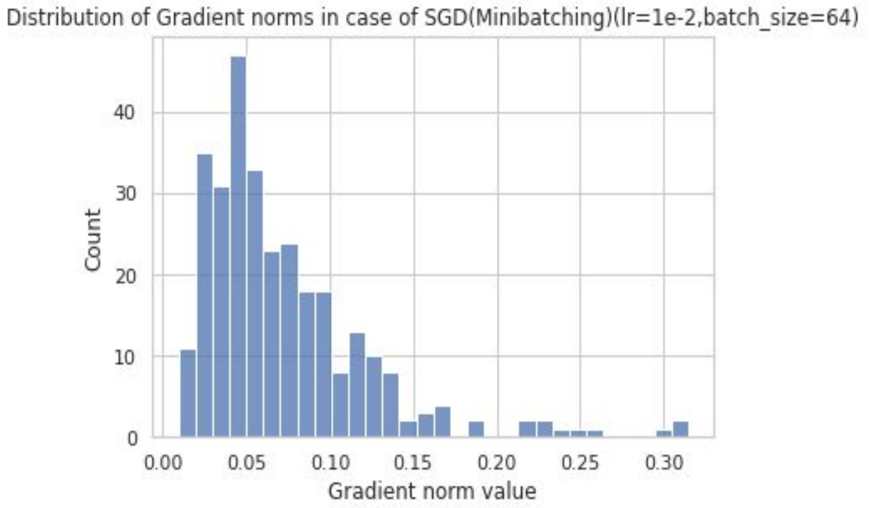




In SGD with mini-batching instead of a single observation, we take a batch of a random sample of fixed size to calculate the gradient. Statistically we know that a batch of sample gives us a better estimate of a parameter than a single observation. In the diagrams it is clear that if we increase the batch size, the loss progression is way more smoother. Also, both the batch sizes give us smoother loss progression than vanilla SGD

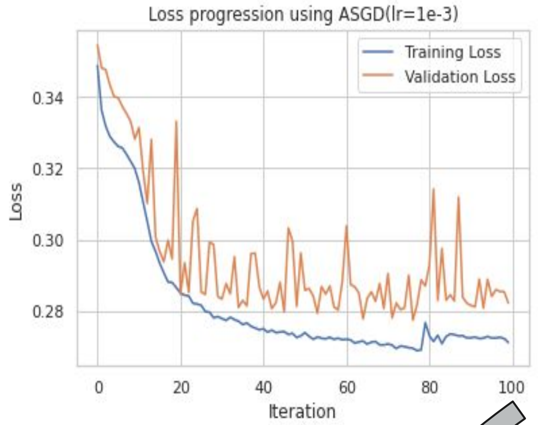
SGD WITH MINI-BATCHING (Continued…)

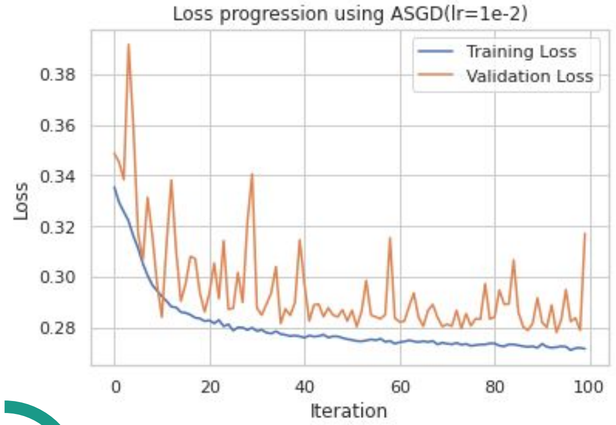


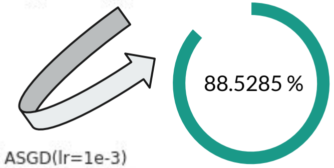


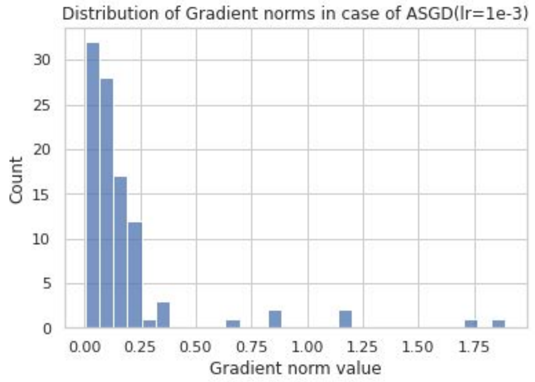
When we increase the sample size or batch size, it should estimate the actual true gradient in a better manner. From the histograms above, we can observe that the variance of estimated gradient has decreased, though neither of them gives us the unbiased estimate of the true gradient, because the distribution is multimodal and they neither show maximum concentration at zero nor show a steep diminish after zero.

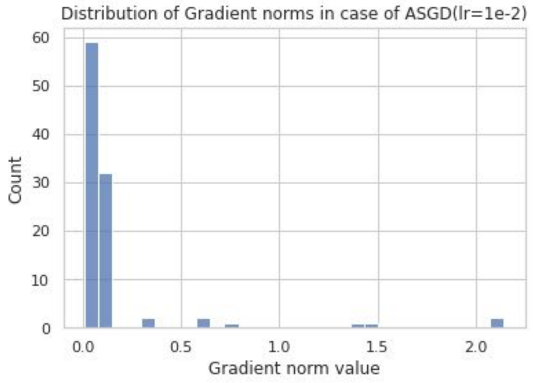
AVERAGE STOCHASTIC GRADIENT DESCENT (ASGD)











The averaging is used to reduce the effect of noise. Namely, in practice, your gradient descent may get close to the optimal, but not really converge to it, instead oscillating around the optimal. In this case, averaging the results of stochastic gradient descent will give you a solution more likely to be closer to the optimum.

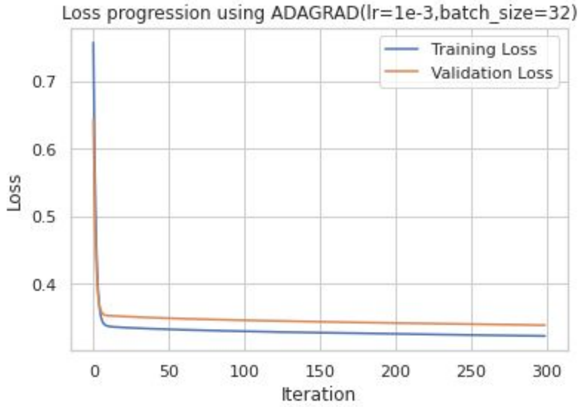
The basic idea is to use the SGD update rule,

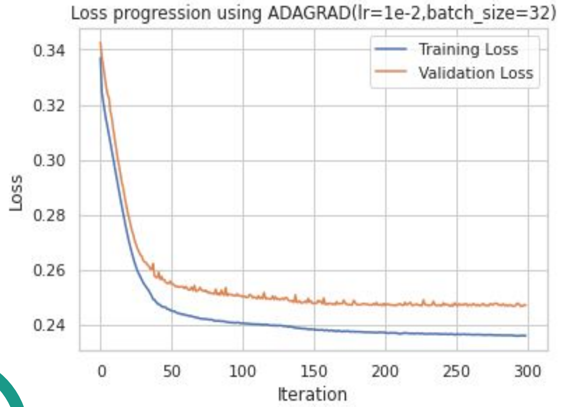
$$w := w - \eta \nabla Q_i(w).$$

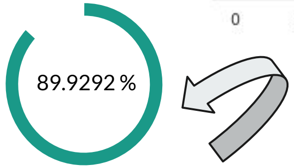
and take the output as the average output of SGD,

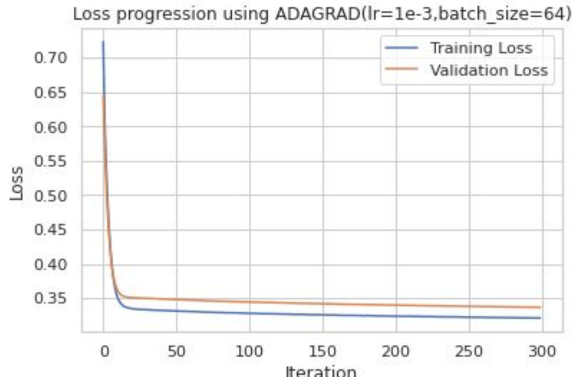
$$\bar{w} = \frac{1}{t} \sum_{i=0}^{t-1} w_i$$

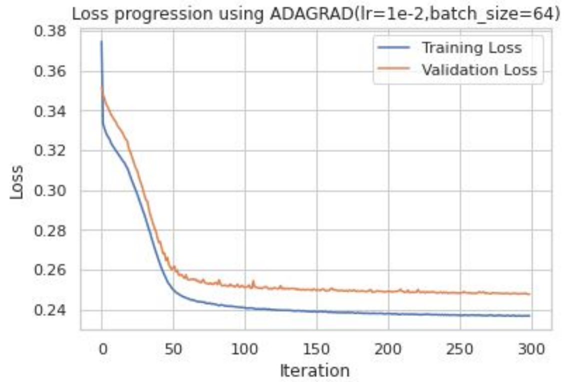
ADAPTIVE GRADIENT ALGORITHM (AdaGrad)





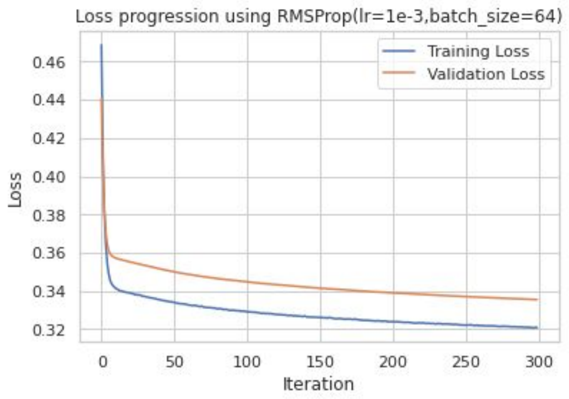


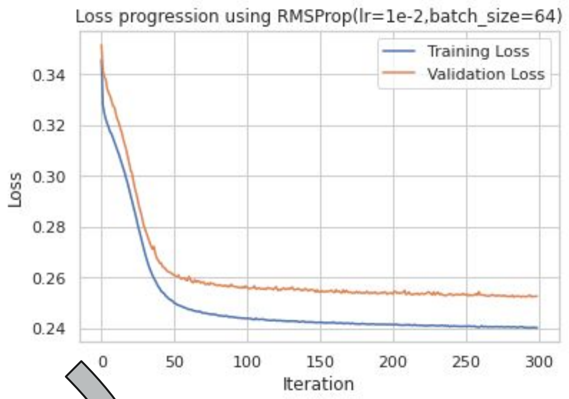


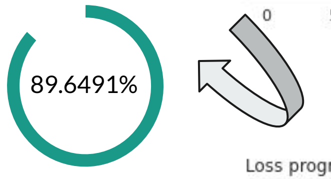


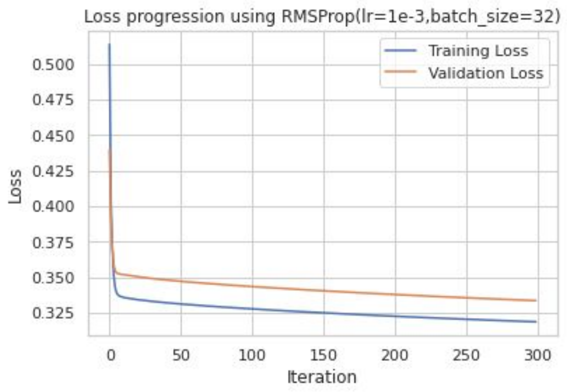
Adaptive Gradient Algorithm (Adagrad) is an algorithm for gradient-based optimization. The learning rate is adapted component-wise to the parameters by incorporating knowledge of past observations. It performs larger updates (e.g. high learning rates) for those parameters that are related to infrequent features and smaller updates (i.e. low learning rates) for frequent one. It performs smaller updates As a result, it is well-suited when dealing with sparse data (NLP or image recognition) Each parameter has its own learning rate that improves performance on problems with sparse gradients and also handles the vanishing or exploding gradient problem.

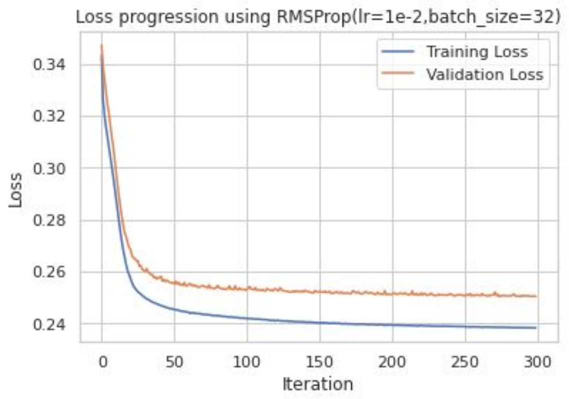
ROOT MEAN SQUARE PROPAGATION (RMSProp)





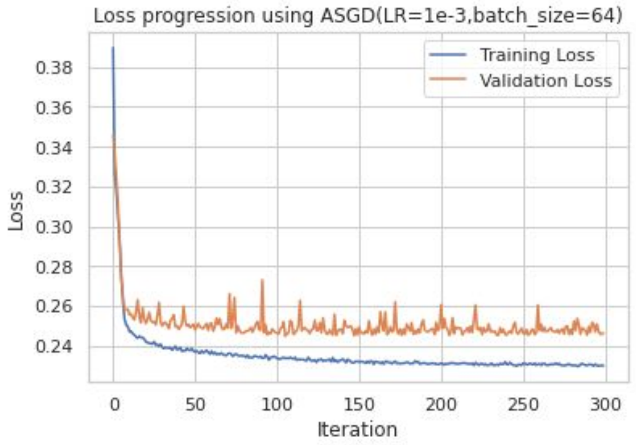


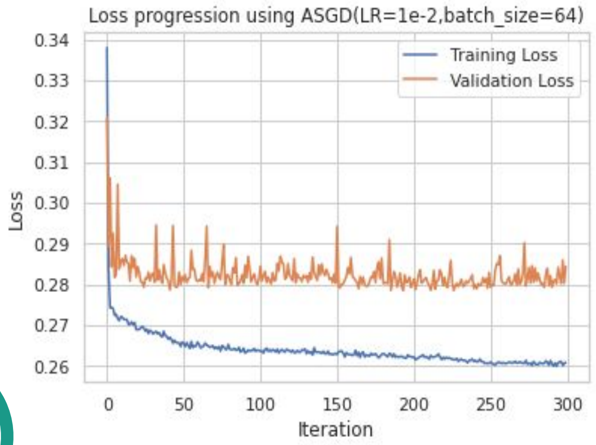


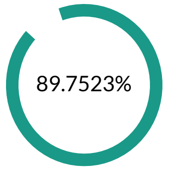


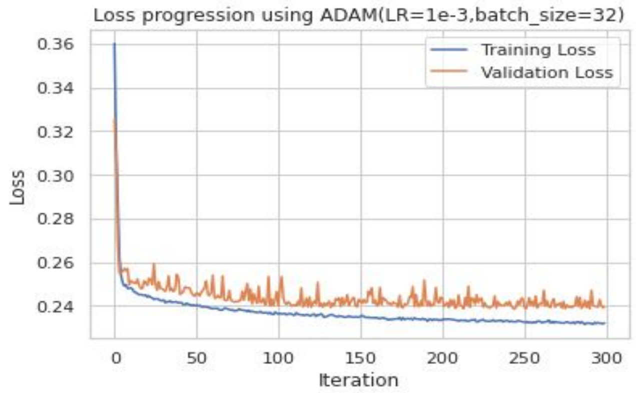
RMSProp is an extended version of AdaGrad. It deals with the vanishing gradient issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. This means that the learning rate changes over time.

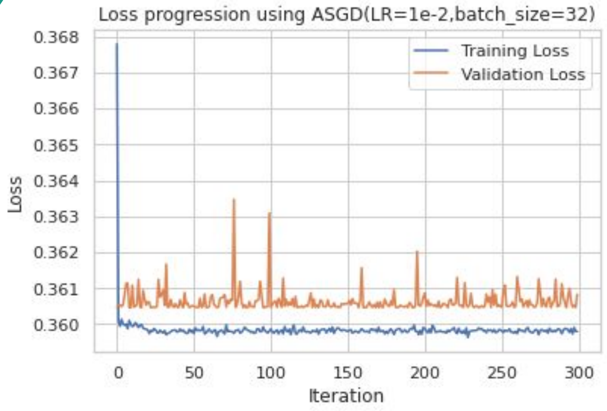
ADAPTIVE MOMENTUM ESTIMATION (ADAM)











Adam is nothing but a combination of SGD with momentum and RMSProp. It performs very good in case of sparse data or when the problem involves lot of parameters or data instances. It uses momentum and also root mean square propagation and the update rule is given by,

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta w_t}\right]$$

$$+$$

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right]$$

$$v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

$$\Downarrow$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

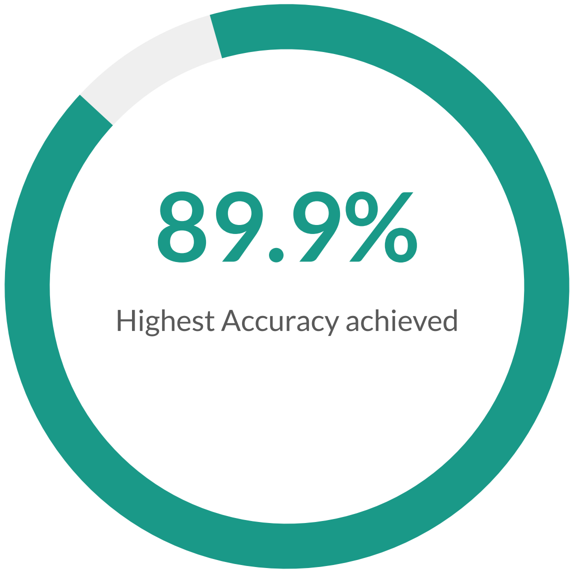
RESULTS

ALGORITHM - AdaGrad

HYPER-PARAMETERS—

Learning rate = 1e-2

Batch size = 64


Highest Accuracy achieved