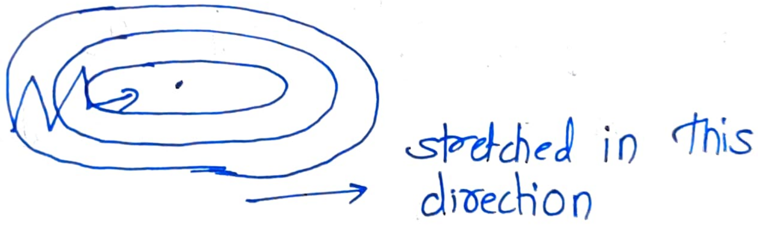
Batch normalization

Batch normalization is an algorithmic method which makes the training of deep neural networks faster and more stable. It consists of normalizing activation vectors from hidden layer using the mean and variance of the current batch. This normalization step is applied right before(or right after) the non linear function.

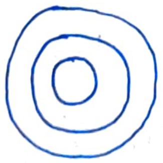
Why use batch norm?

Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape. The reason we normalize is partly to ensure that our model can generalize appropriately.

-Input should be on same scale due to which mean becomes 0 and standard deviation becomes 1 for all the input data. If data is not normalized, plot looks like,



-After data is normalized, data plot look like, where training is faster and stable.

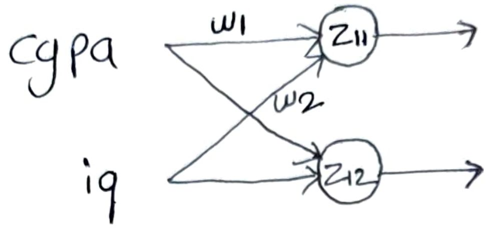


- Hence output of activation function also should be normalized.

-Internal Covariate Shift(explained at the end)

How it works?

-Batch normalization is done individually at each unit.



$z_{11} = w_1 cgpa + w_2 iq + b$

- Consider batch size $i=4$
- Now Calculate the mean & standard deviation for this batch.

$$\mu_\beta = \frac{1}{m} \sum_{i=1}^{m} z_{11}^{i} \qquad \text{here } m=4$$

$$\sigma_\beta = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (z_{11}^{i} - \mu)}$$

This is for z_{11}. Same Calculate for z_{12}

$$\therefore \quad z_{11}^{N} = \frac{z_{11}^{i} - \mu_\beta}{\sigma_\beta + \varepsilon}$$

For keras, $\gamma=1$ & $\beta=0$ at initial step.

Now multiply z_{11}^{N} with γ (gamma) & add β.
γ, & β are learnable parameters.

$$\therefore \quad \boxed{z_{11}(\text{final}) = \gamma z_{11}^{N} + \beta}$$

Internal Covariate Shift

Let's say we have a flower dataset and want to build a binary classifier for roses. The output is 1 if the image is that of a rose, and the output is 0 otherwise.

Consider a subset of the training data that primarily has red rose buds as rose and wildflowers as non-rose examples. These are shown in Figure 1.

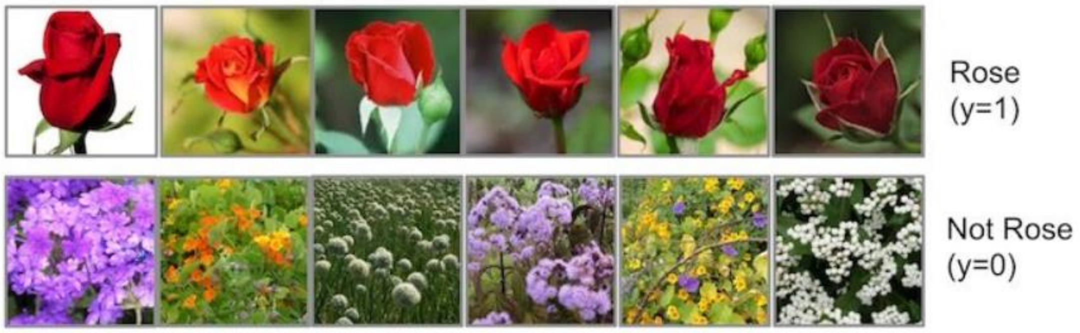


Figure 1

Consider another subset, shown in Figure 2, that has fully blown roses of different colors as rose examples and other non-rose flowers in the picture as non-rose examples.

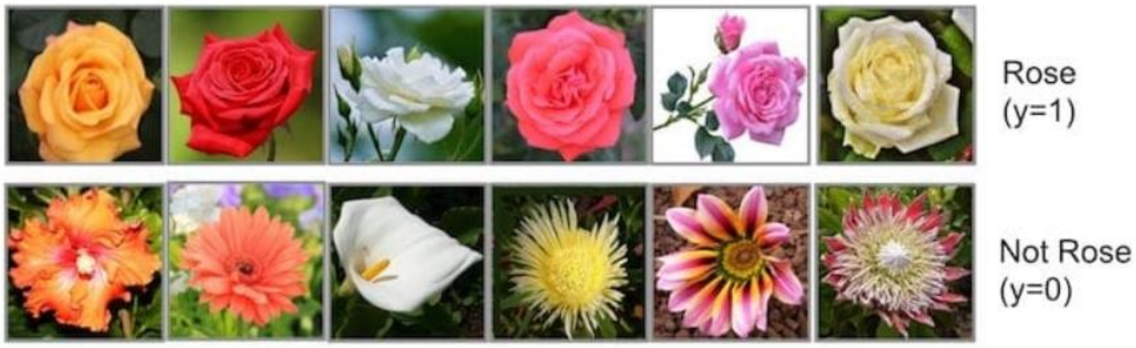


Figure 2

It makes sense that every mini-batch used in the training process should have the same distribution. In other words, a mini-batch should not have only images from one of the two subsets above. It should have images randomly selected from both subsets in each mini-batch.

The same intuition is graphically depicted in Figure 3. The last column of Figure 3 shows the two classes (roses and non-roses) in the feature space (shown in two dimensions for ease of visualization). The blue curve shows the decision boundary. We can see the two subsets lie in different regions of the feature space. This difference in distribution is called the covariate shift. When the mini-batches have images uniformly sampled from the entire distribution, there is negligible covariate shift. However, when the mini-batches are sampled from only one of the two subsets shown in Figure 1 and Figure 2, there is a significant covariate shift. This makes the training of the rose vs non-rose classifier very slow.

An easy way to solve this

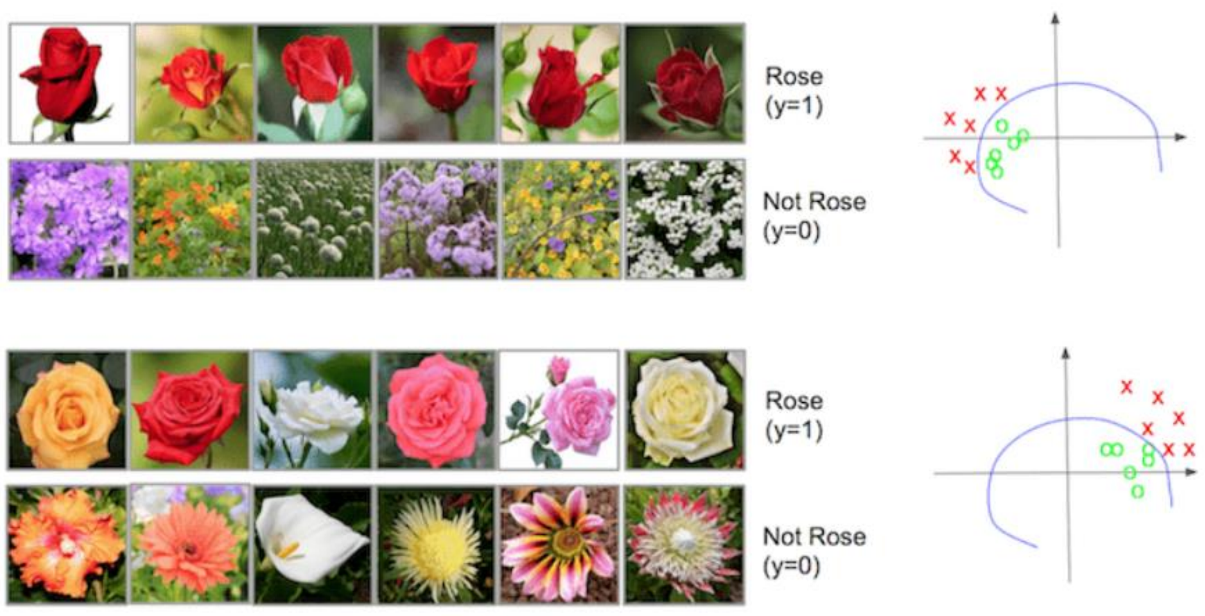


Figure 3: The top two rows of flowers show a subset of the data and the bottom two rows show a different subset of the data. The two subsets have very different distributions. The last column shows the distribution of the two classes in the feature space using red and green dots. The blue line show the decision boundary between the two classes.

Just as it made intuitive sense to have a uniform distribution for the input layer, it is advantageous to have the same input distribution for each hidden unit over time while training. But in a neural network, each hidden unit's input distribution changes every time there is a parameter update in the previous layer. This is called internal covariate shift. This makes training slow and requires a minimal learning rate and a good parameter initialization. This problem is solved by normalizing the layer's inputs over a mini-batch, which is called Batch Normalization.