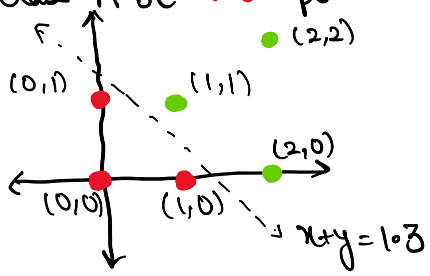
ML Assignment Section A

Monday, 14 November 2022 2:55 PM

① Let class A be red points & B be green points



we see that the line $y+x=1.3$ correctly classifies all points as the red points lie on one side while the other green points are on the other side

② Let $\beta_1=(0,1)$, $\beta_2=(1,0)$, $\beta_3=(1,1)$, $\beta_4=(2,0)$, $\beta_5=(0,0)$, $\beta_6=(2,2)$

To maximize margin $\frac{\|\omega\|^2}{2}$ with the constraint $y_i[\omega^T x_i+b_i]$

$y_1=-1$, $y_2=-1$, $y_3=1$, $y_4=1$, $y_5=-1$, $y_6=1$

We maximize,

$$J(\omega, b, \alpha) = \frac{\|\omega\|^2}{2} - \sum_{i=1}^{6} \alpha_i [y_i(\omega^T x_i + b)]$$

No line passing via $(2,2)$ & $(0,0)$ can correctly classify the data.

Hence $\alpha_5=\alpha_6=0$

So now our loop only goes for 4 iterations

So our dual form which we maximize is

$$Q(\alpha) = \sum_{i=1}^{4}\alpha_i - \frac{1}{2}\left[\sum_{i=1}^{4}\sum_{j=1}^{4}\alpha_i\alpha_j y_i y_j \beta_i\beta_j\right]$$

Subject to the condition $\sum_{i=1}^{4}\alpha_i y_i = 0$ & $\alpha_i \geq 0 \ \forall i$

Now, $\sum_{i=1}^{4}\alpha_i y_i = 0 \Rightarrow \alpha_1+\alpha_2=\alpha_3+\alpha_4$

$$Q(\alpha) = (\alpha_1+\alpha_2+\alpha_3+\alpha_4) - \frac{1}{2}[\alpha_1^2+\alpha_2^2+2\alpha_3^2+4\alpha_4^2 - 2\alpha_1\alpha_3 - 2\alpha_2\alpha_3 - 4\alpha_2\alpha_4 + 4\alpha_3\alpha_4]$$

Now we use $\alpha_1 = \alpha_3+\alpha_4-\alpha_2$

$\Rightarrow -2\alpha_1\alpha_3 = (-2\alpha_3^2 - 2\alpha_3\alpha_4 + 2\alpha_2\alpha_3)$

So $Q(\alpha)$ will now become

$$Q(\alpha) = \alpha_1+\alpha_2+\alpha_3+\alpha_4 - \frac{1}{2}[\alpha_1^2+\alpha_2^2-4\alpha_4^2 - 4\alpha_2\alpha_4 + 2\alpha_3\alpha_4]$$

$\frac{\partial Q(\alpha)}{\partial \alpha_1} = 0 \Rightarrow 1-\alpha_1 = 0$

$\Rightarrow \alpha_1 = 1$

$\frac{\partial Q(\alpha)}{\partial \alpha_2} = 0 \Rightarrow 1-\frac{1}{2}[2\alpha_2-4\alpha_4]$

$\Rightarrow \alpha_2 - 2\alpha_4 = 1$

$\frac{\partial Q(\alpha)}{\partial \alpha_3} = 0 \Rightarrow 1-\frac{1}{2}[8\alpha_4-4\alpha_2+2\alpha_3] = 0$

$\Rightarrow \alpha_3 = 3$

$$\omega = \sum_{i=1}^{4}\alpha_i y_i \beta_i = -1\begin{bmatrix}0\\1\end{bmatrix} + (-3)\begin{bmatrix}1\\0\end{bmatrix} + 3\begin{bmatrix}1\\1\end{bmatrix} + 1\begin{bmatrix}2\\0\end{bmatrix}$$

[2 ...]

$$\omega = \sum_{i=1} \alpha_i y_i \beta_i = -1\begin{bmatrix}0\\1\end{bmatrix} + (-6)\begin{bmatrix}1\\0\end{bmatrix} + 5\begin{bmatrix}1\\1\end{bmatrix} + 1\begin{bmatrix}2\\0\end{bmatrix}$$
$$= \begin{bmatrix}2\\2\end{bmatrix}$$

$b = 1 - \omega^T \beta_i$ s.t. $y_i = 1$

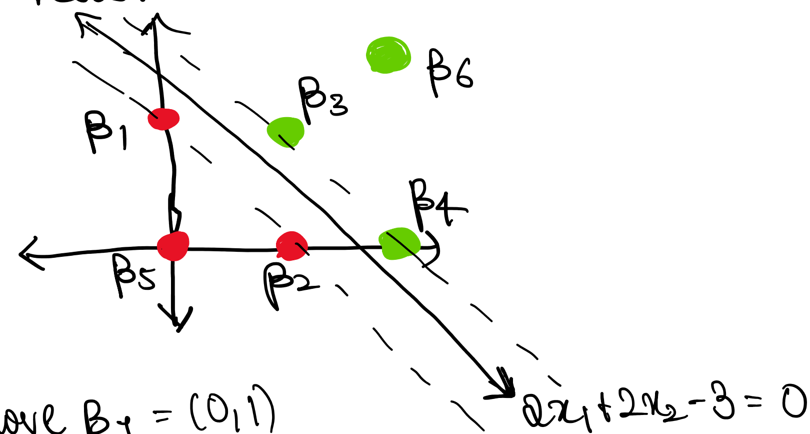
Let $\beta_i = \beta_3$

$b = 1 - \begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = 1 - 4 = -3$

$\therefore$ optimal line is $2x_1 + 2x_2 - 3 = 0$

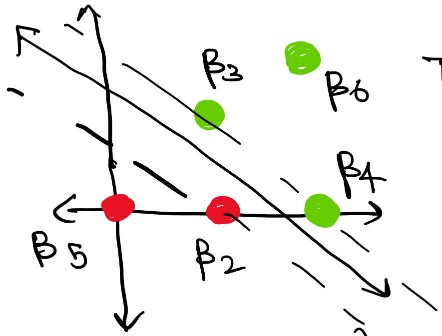
Length of optimal margin $= \dfrac{2}{\|\omega\|} = \dfrac{2}{\sqrt{2^2+2^2}} = \dfrac{2}{2\sqrt{2}} = \dfrac{1}{\sqrt{2}}$

No α_i for $i = 1,2,3,4$ is 0 so all of them are support vectors



(3) Case I → Remove $\beta_1 = (0,1)$

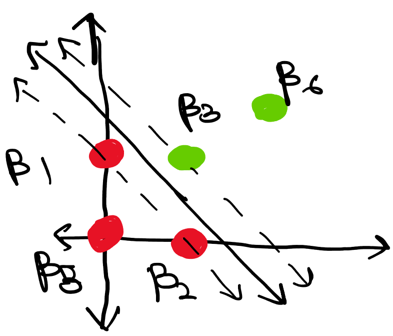
NO CHANGE



The support vectors are now $(1,0)$, $(1,1)$, $(2,0)$

Case II → Remove $\beta_4 = (2,0)$.

NO CHANGE



The support vectors are now $(1,0)$, $(0,1)$, $(1,1)$

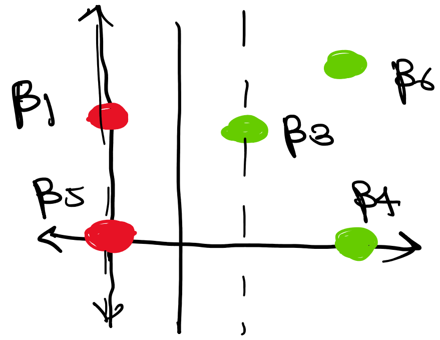
Case III → Remove $\beta_2 = (1,0)$

MARGIN CHANGES

NEW MARGIN HAS TOTAL LENGTH = 1

The support vectors are

NEW MARGIN HAS TOTAL LENGTH = 1

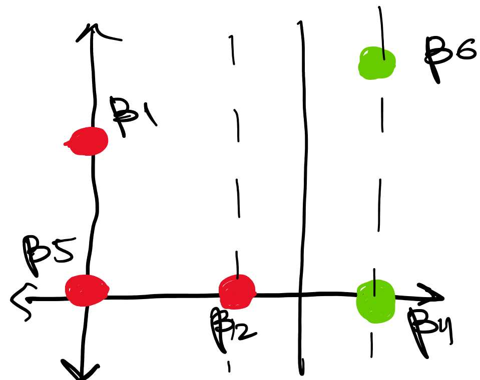


The support vectors are now (0,0), (0,1), (1,1)

Case IV → Remove $\beta_3 = (1,1)$

MARGIN CHANGES

NEW MARGIN HAS TOTAL LENGTH = 1

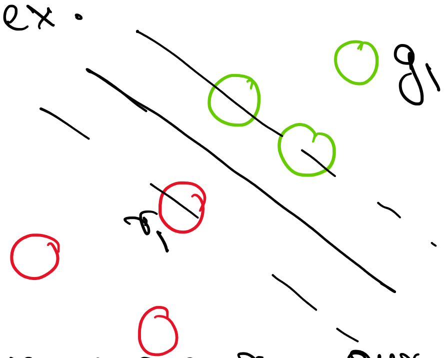


The support vectors are (1,0), (2,0), (2,2)

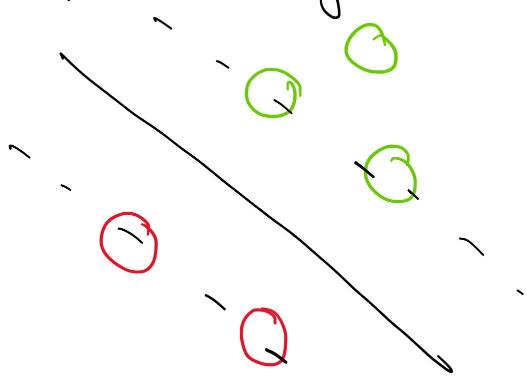
(4) The margin may or may not change as we saw in part (3)

If some new point gets introduced which is a support vector then the margin changes

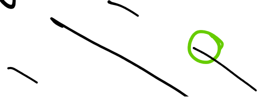
For ex.

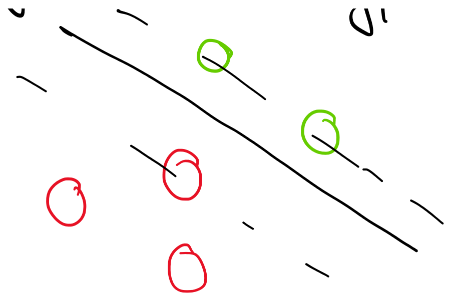


If we remove r_1 our margin and support vectors change



If we remove g_1 there is no change in margin





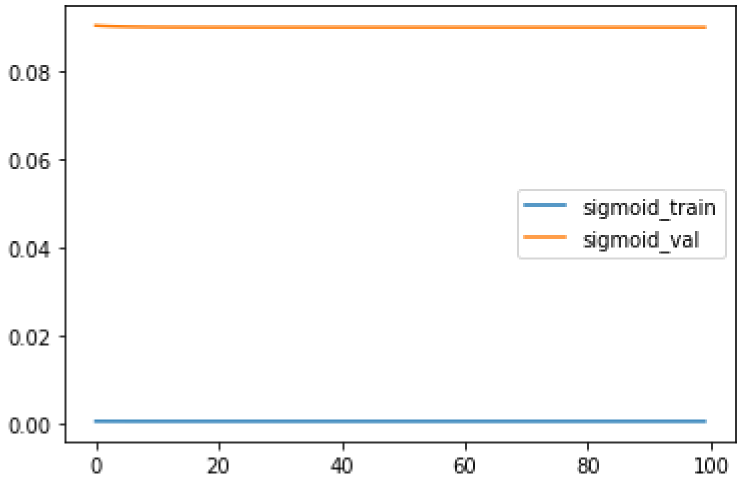
Report

Section B

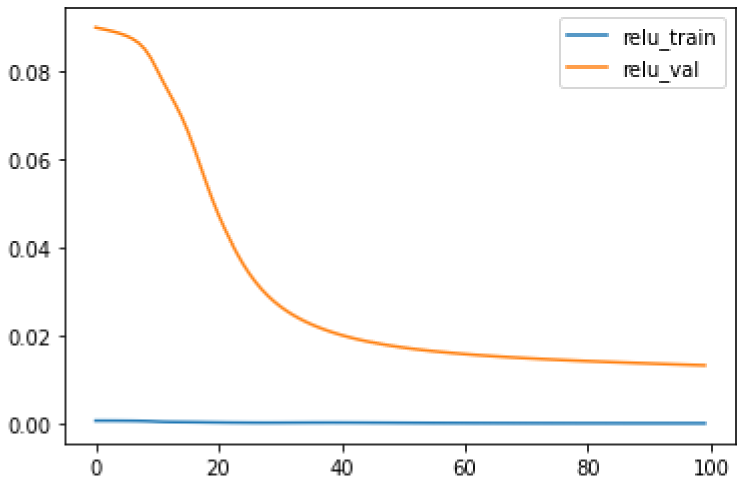
I've implemented Early Stopping Using Validation Set. Since the dataset was too large only 8K Training and 2K Validation Samples are Used. The Training is set for a maximum of 100 epochs.

Activation Function Testing:

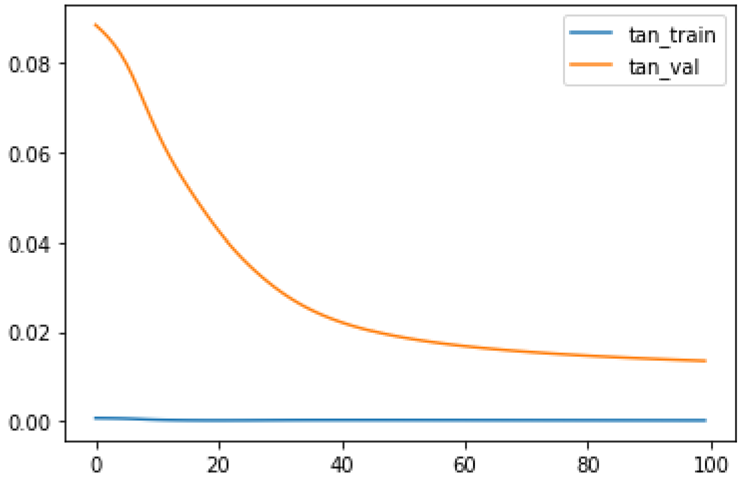
Sigmoid: Accuracy: 0.1135



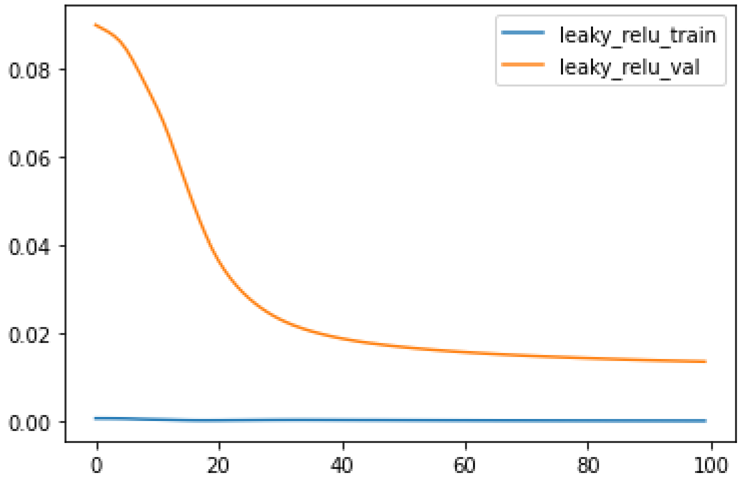
Relu: Accuracy: 0.9163



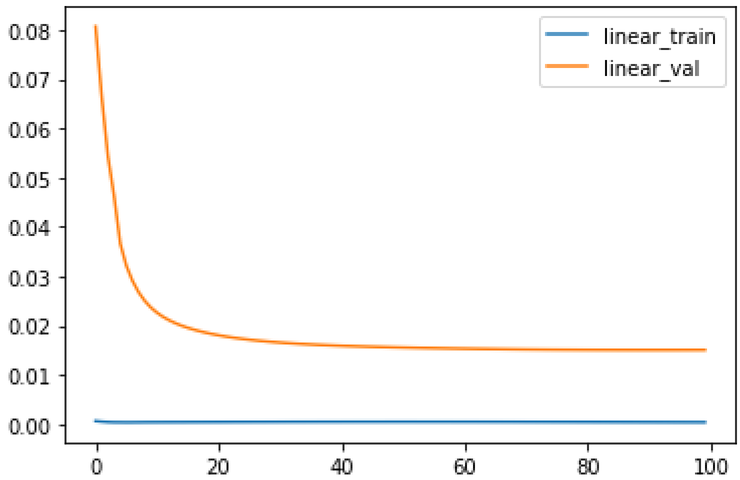
Tanh: Accuracy: 0.9193



Leaky Relu: Accuracy: 0.9138



Linear: Accuracy: 0.9056



Hence the Best Activation Function is – Tanh

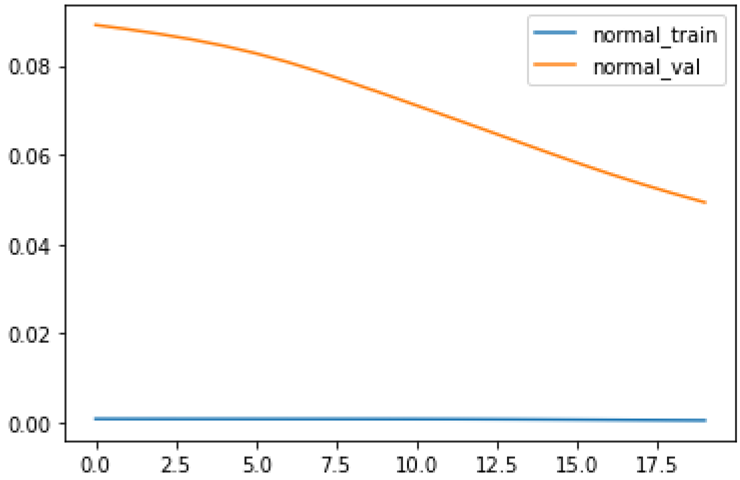
However, Tanh, relu and leaky relu all 3 are fairly close.

Suprisingly Sigmoid performs very poorly which might be because of the poor suitability of sigmoid for this task as compared to other activation functions.

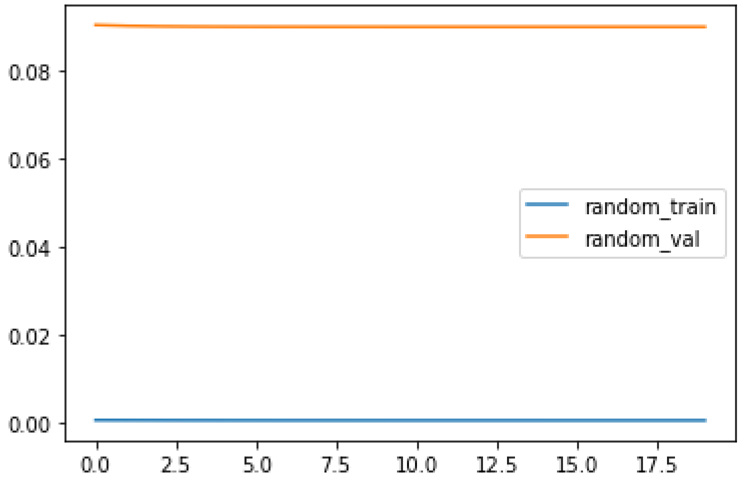
Weight Initialization Testing:

To Compare different Initializations, I only run the model for 20 epochs to see signs of convergence and see which initialization works better

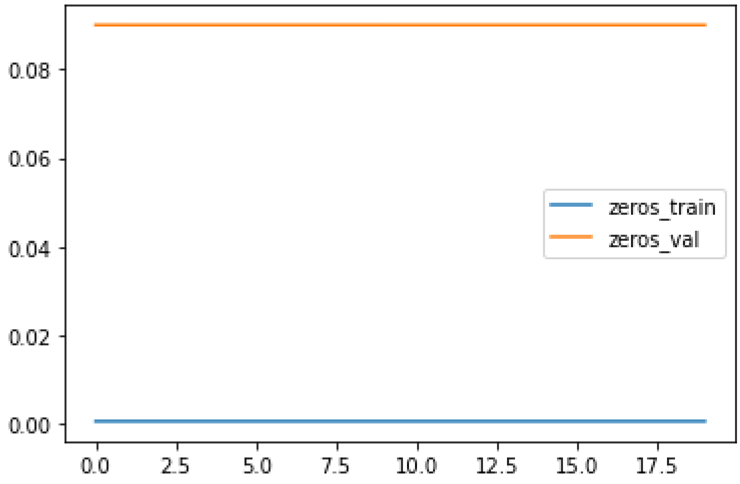
Normal Init: Accuracy: 0.6699



Random Init: 0.1135



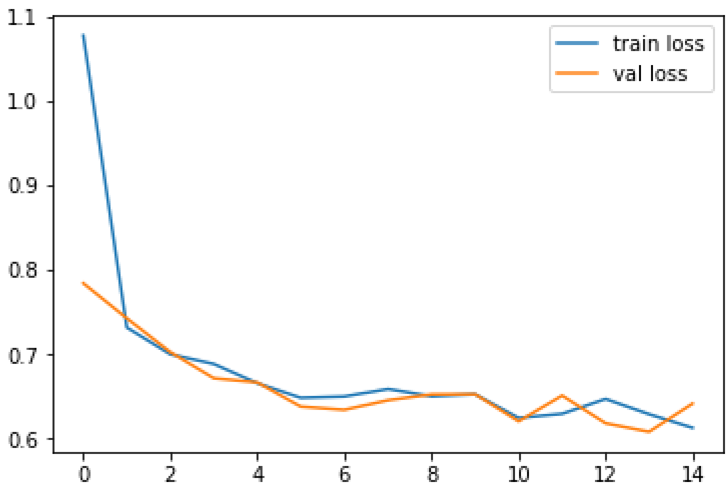
Zeroes: Accuracy: 0.1135



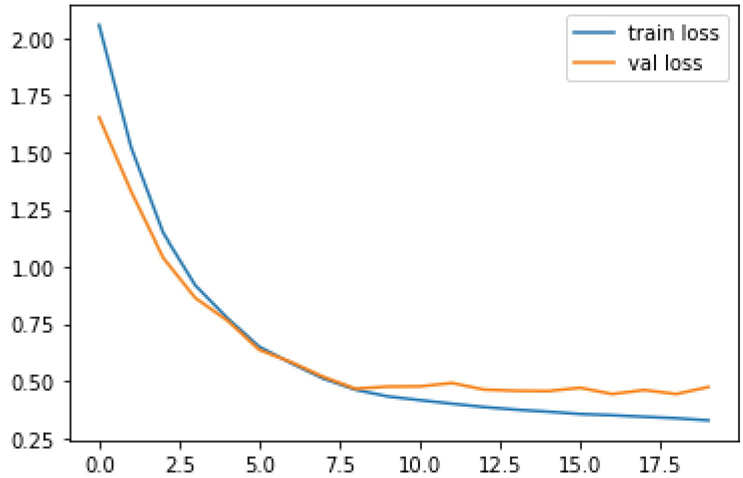
Zeroes Initialization and random initialization work poorly. For the first one as the model updates are similar and it fails to learn more features. For the second one randomness can be unfavourable and hence it can and cannot work depending on the case.

Section C

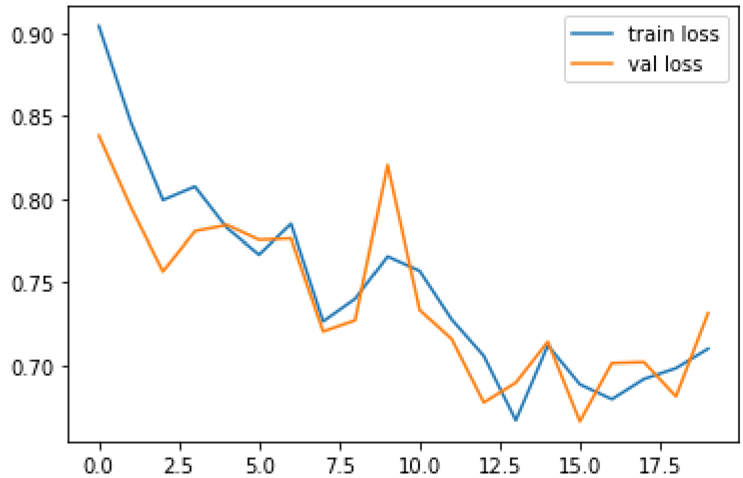
1) For Sigmoid:



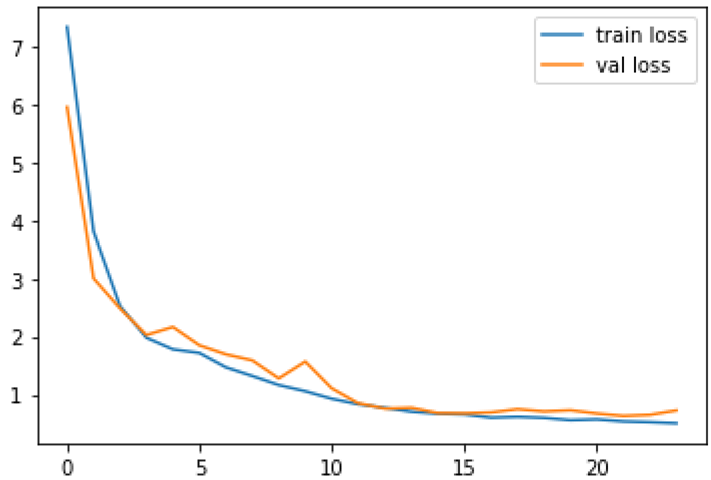
For RELU:



For Tanh:



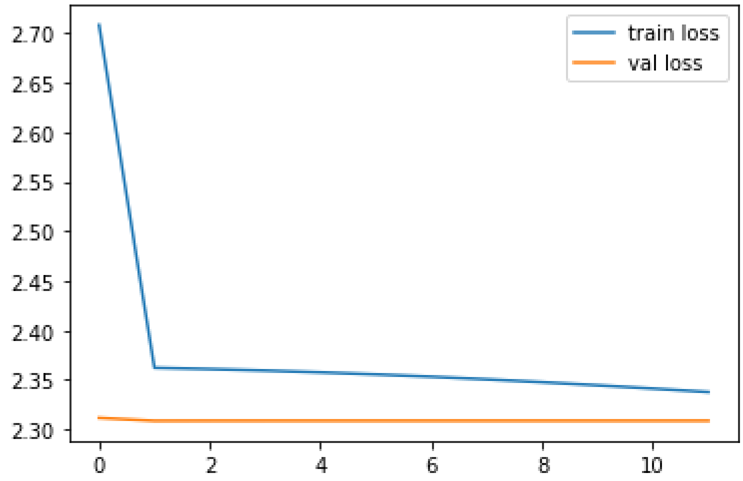
For Linear:



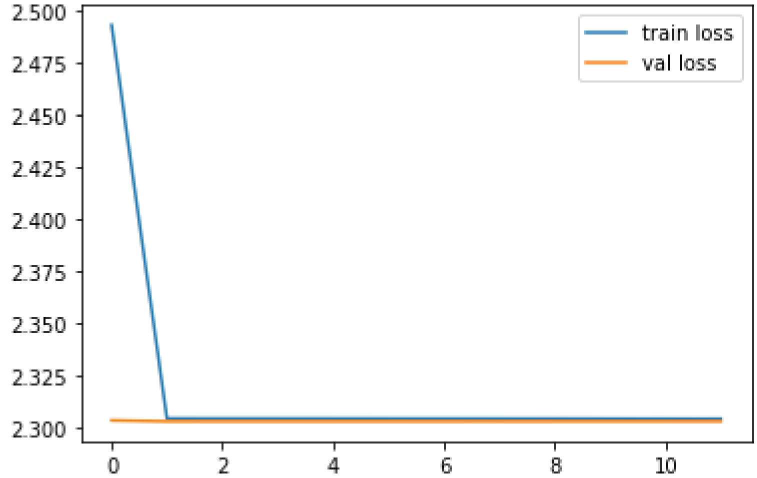
Activation Function	Accuracy
Sigmoid	0.742
Relu	0.852
Tanh	0.7083
Linear	0.7959

Hence Relu has the best accuracy and has a smooth loss curve indicating it's the best choice. The main reason Relu performs so well is that it doesn't activate all neurons at the same time and hence it selects some neurons to activate making the other neurons compensate and learn.

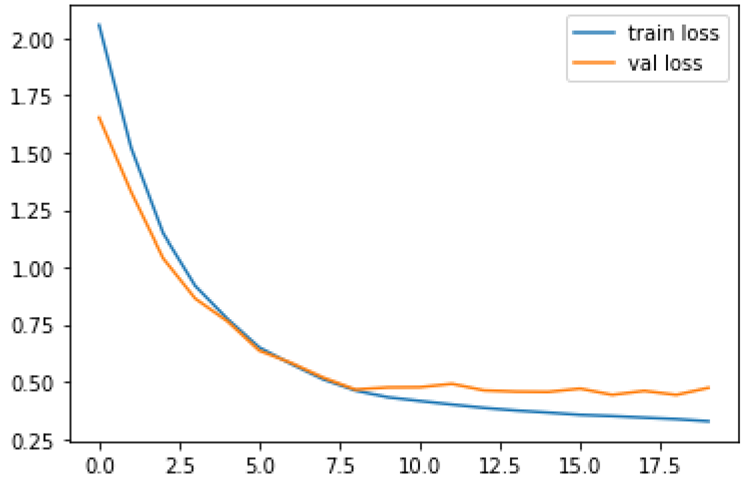
2) Learning Rate: 0.1



Learning Rate: 0.01



Learning Rate: 0.001

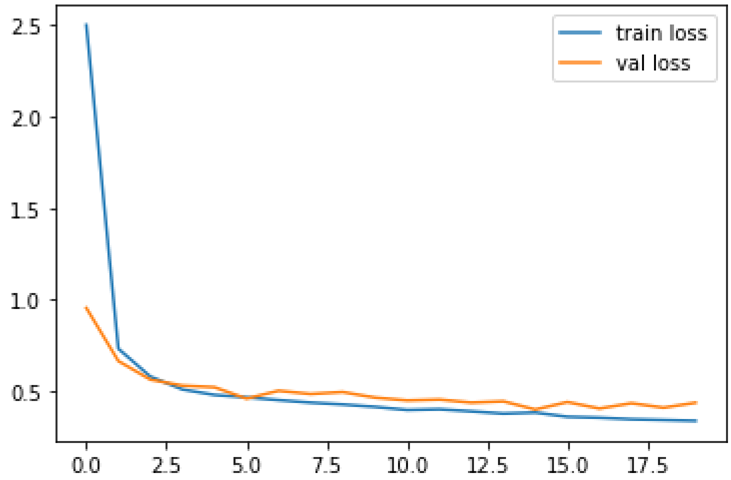


Learning Rate	Accuracy

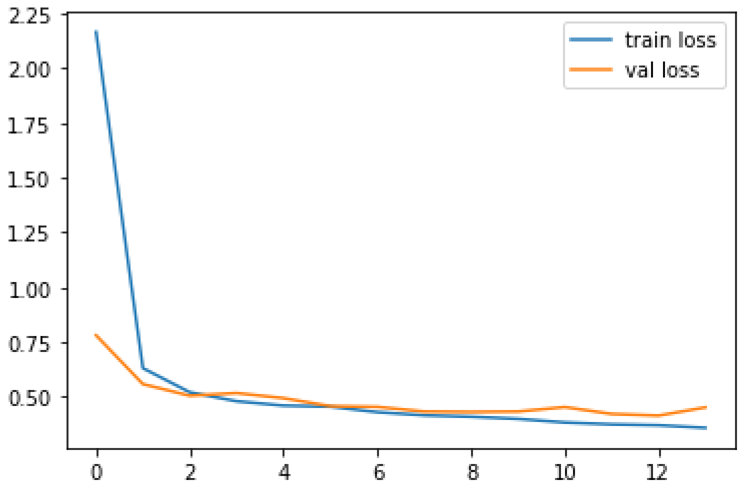
0.1	0.1
0.01	0.1
0.001	0.8512

Hence 0.001 is the best learning rate. The other learning rates fail to converge. Their graphs also have sharp turns. This implies that these learning rates are too high we might take too large of a step and cross the minima

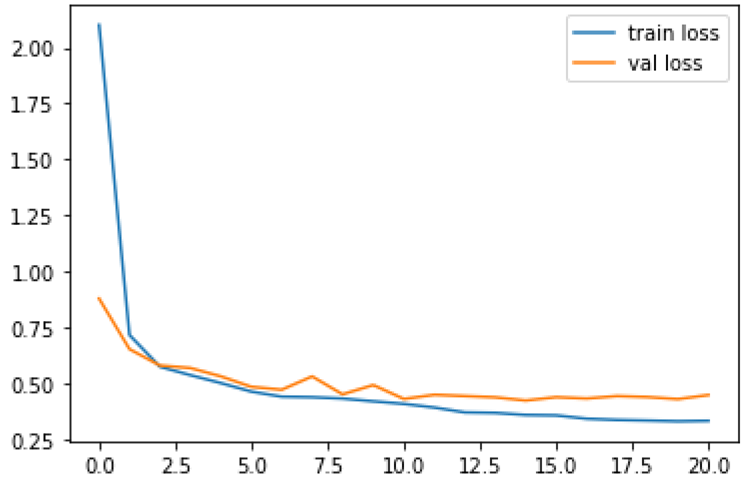
3) (64,64) Neurons



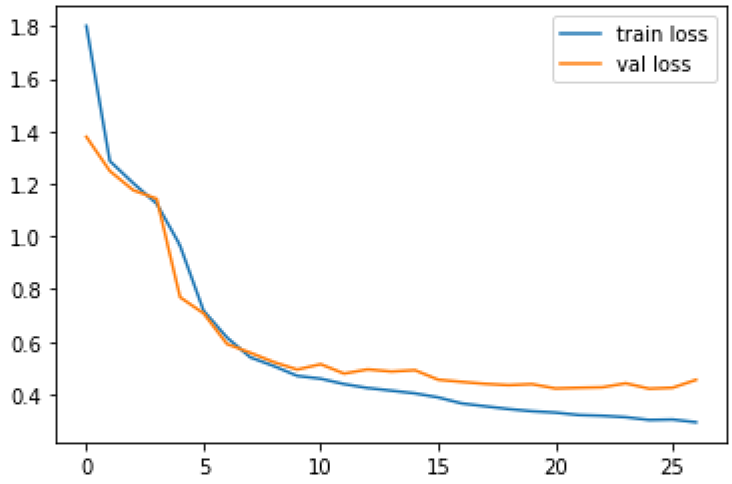
(128, 64) Neurons



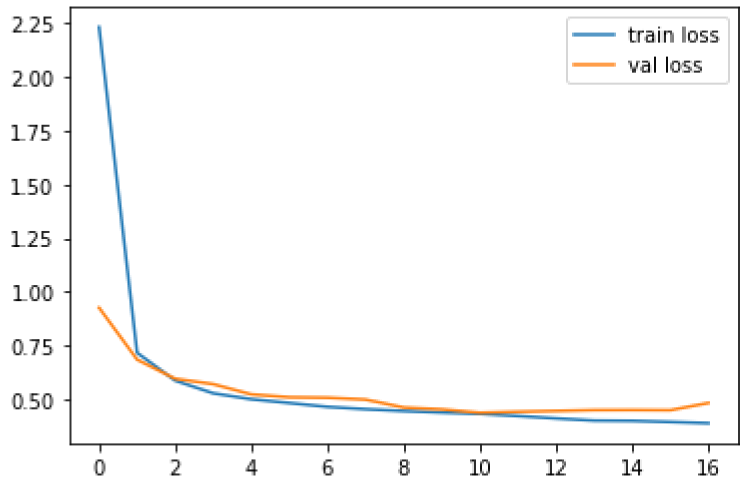
(128, 128) Neurons



(128, 32) Neurons



(64,32) Neurons



Hidden Layer Sizes	Accuracies
(64,64)	0.8603
(128,64)	0.8405
(128,128)	0.85
(128,32)	0.8533
(64,32)	0.8336

As we can see by the accuracies the hidden dimensions (128,32) works best

4) After Performing Grid Search the Best Model is –

   'activation': 'relu',

   'batch_size': 32,

   'early_stopping': True,

   'hidden_layer_sizes': (128, 64),

   'learning_rate_init': 0.0001,

   'random_state': 1,

   'solver': 'adam',

   'validation_fraction': 0.15

It gives the accuracy 0.8751 which is better than all values seen before. This is also expected as these values performed well independently and together, they perform even better.

The adam solver is often used in DL Architectures and hence due to it’s complex nature it might be performing well.