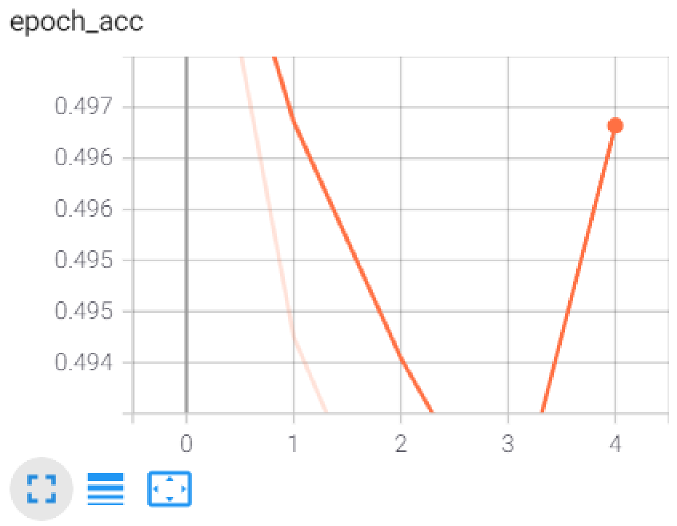
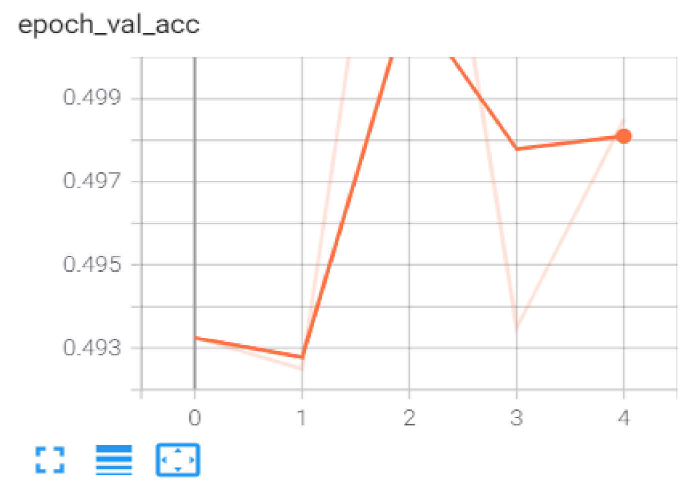
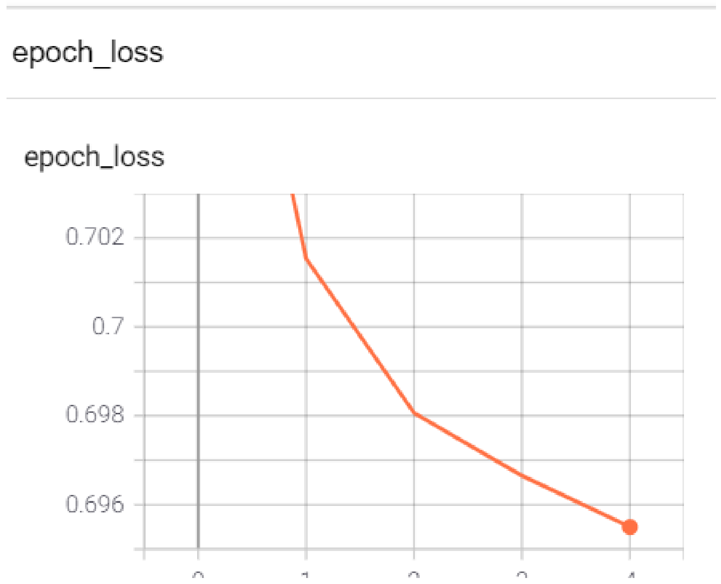
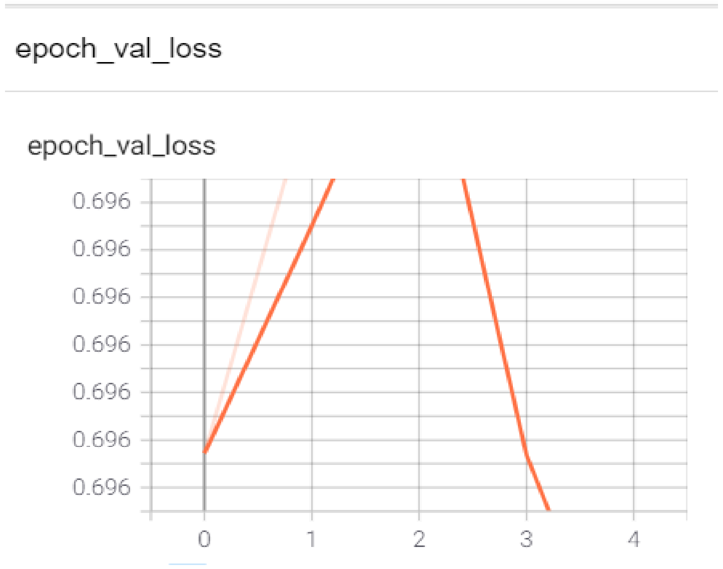
Model 1 Results







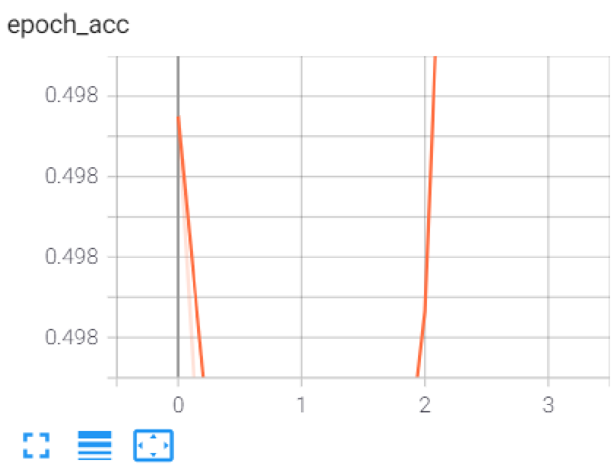


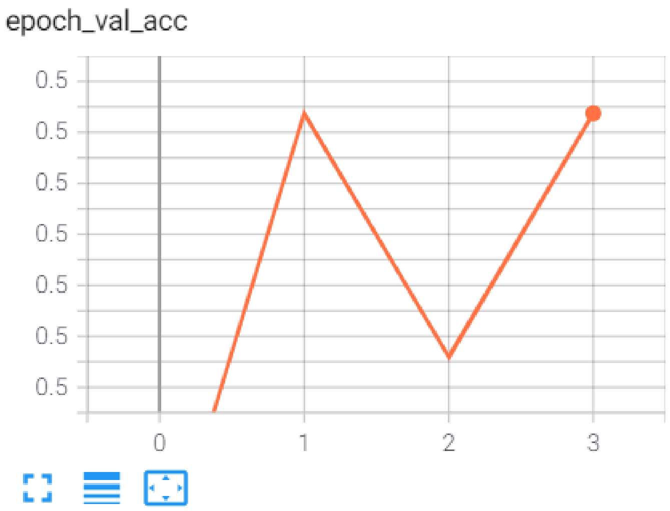
Analysis:

- This model has used ‘tanh’ as the activation unit. The problem with the ‘tanh’ or the sigmoid activation units are te vanishing gradients. As the number of layers increase, the property of chain rule leads to product of excessively small numbers, such that change in gradients becomes so small. If the gradients become small, the weights may not converge at all, even for large number of epochs.
- The weights have been initialized from a random uniform distribution with a small variance. Suppose if the weights are all 0’s or the same number, then all the neurons will compute the same thing.(No point in deep hidden layers), - i.e, same gradient updates happen to all the activation units.

- This is a problem of symmetry. Each neuron must behave differently to learn different aspect of data (similar to GBDT high variance models)
- The advantage here is , the weights are small and not are the same value, as they are initialized from the small variance. Prevents exploding/vanishing gradient problem.
- Tanh/sigmoid unit converges a little slower than relu activation units.
- A step decay learning rate along with momentum SGD is used to relatively speed up the convergence rate.

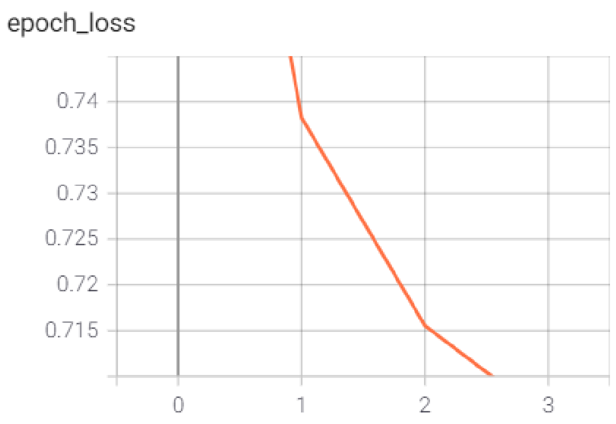
Model 2 Results:

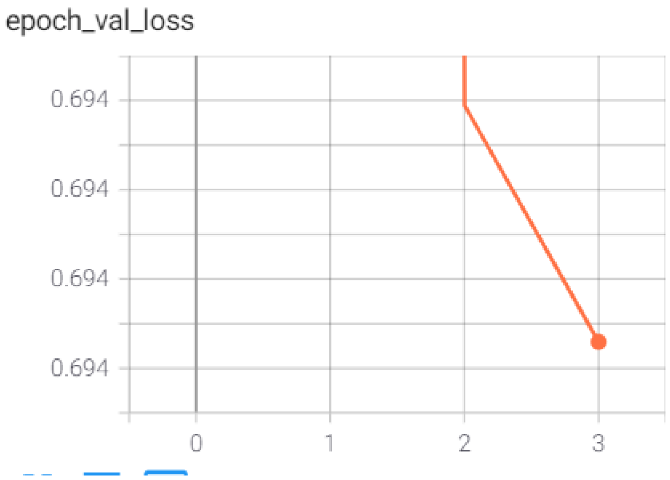




epoch_loss

epoch_val_loss

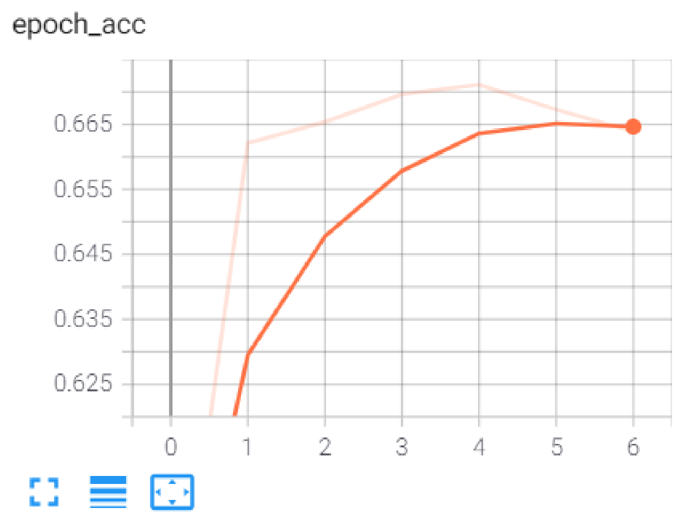


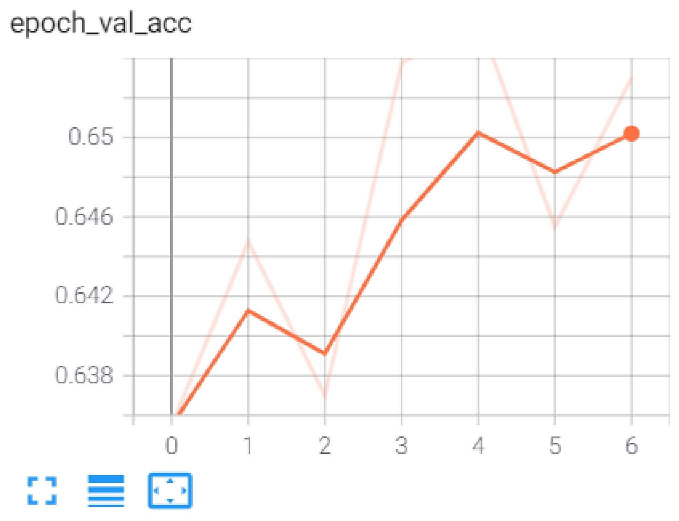


Analysis:

- Val_loss is much lower compared to model 1 'tanh' activation unit, in lesser epochs.
- Val_acc has also improve better than model in lesser epochs.
- Relu activation overcomes the problem of vanishing gradient due to tanh activation, thereby leading to faster convergence.
- The advantage of Relu is that for Z= W'X for Z > 0 the Relu(Z) = Z itself, so computing gradients is computationally cheap and easy to compute.\
- For a really deep neural architecture, Relu speeds up convergence immensely and makes the gradient computation easier.
- But the problem with Relu is , the function is not continuous and hence not differentiable at Z=0. Suppose all the weight components are large and negative, the Relu(Z) =0. The gradients becomes 0 , and no updating takes place for weights. This state is called the dead activation state.
- This problem may be slightly solved by the random uniform weight initialization.

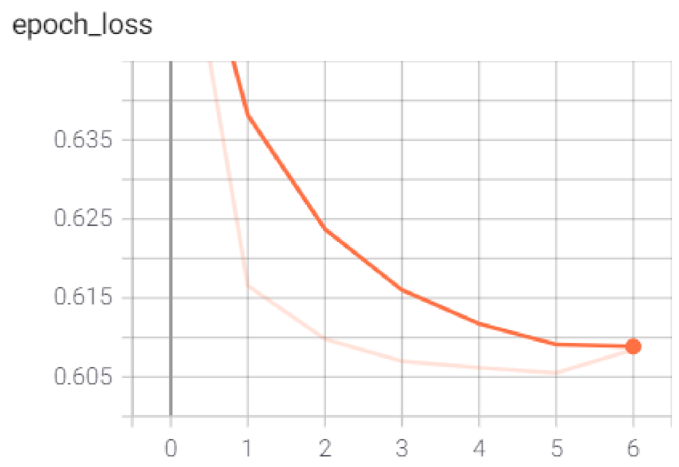
Model 3 Results

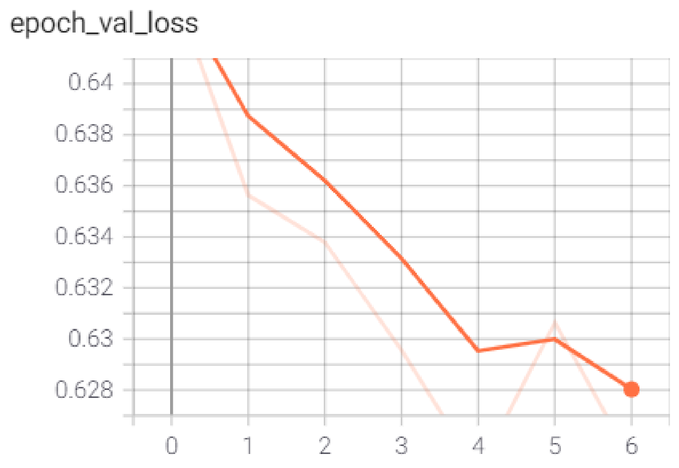




epoch_loss

epoch_val_loss

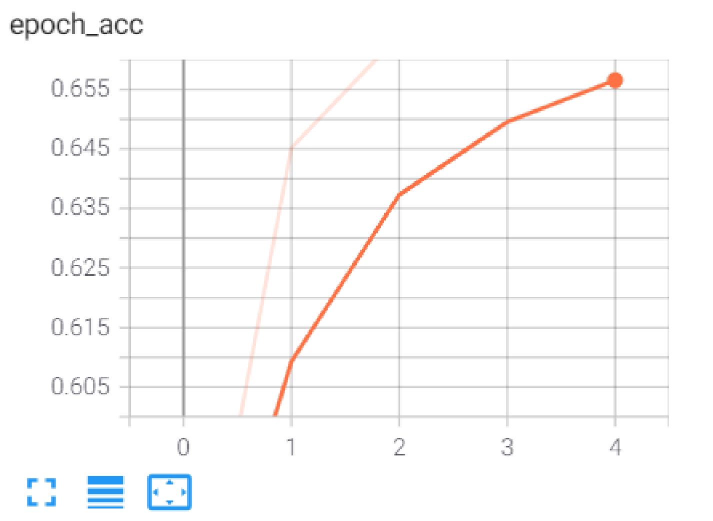


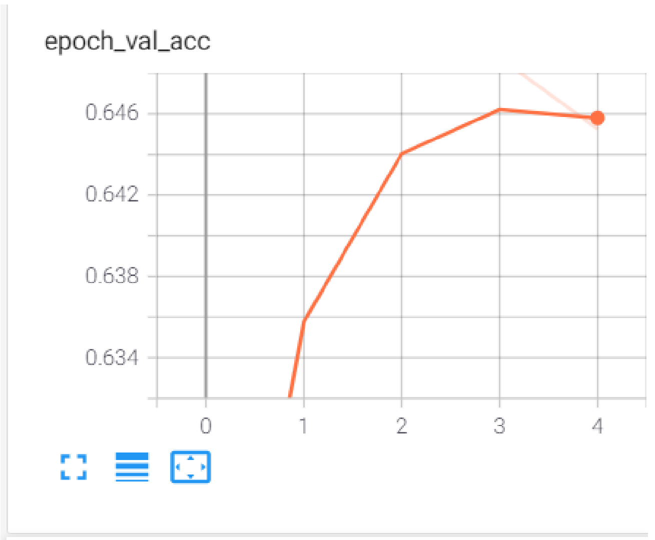


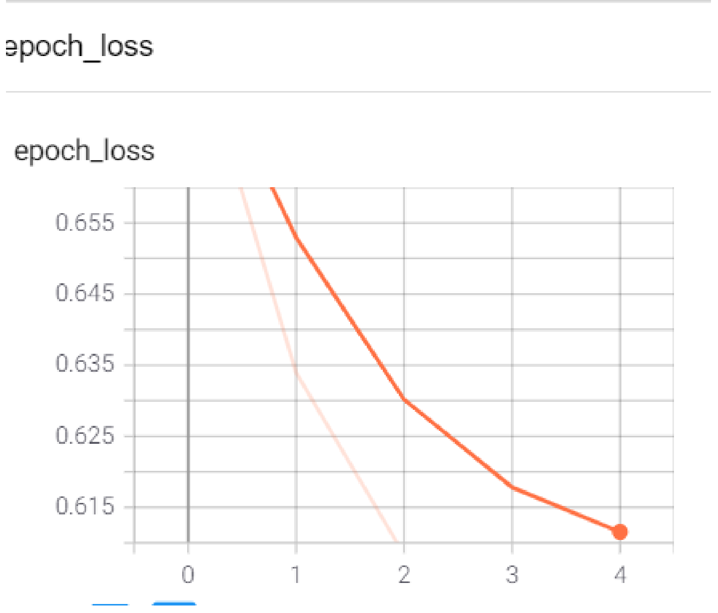
Analysis:

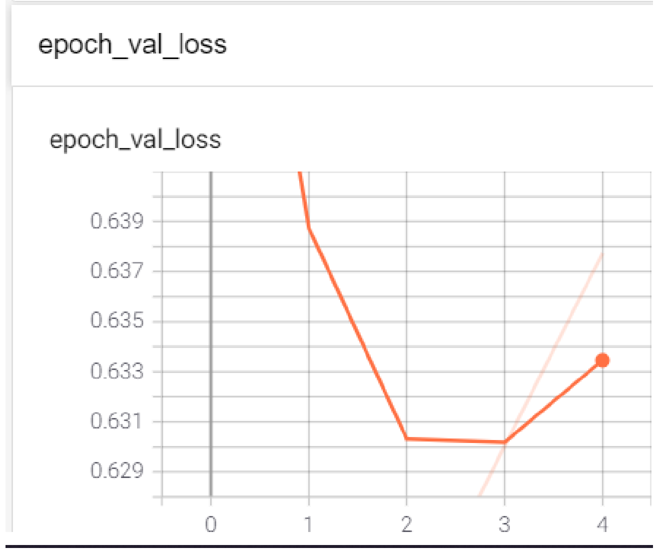
- Val_acc has greatly improved compared to the last model of 0.5 to .65. This could highlight the importance of weight initialization and choosing the optimal activation function.
- Val_loss has also greatly reduced compared to last model of 0.694 with early stopping and 0.628 for 6 epochs.
- The usage of Relu has helped in speeder convergence and initing weights from He_uniform could help prevent the problem of dead activation state. The combined power of both into the model has greater improvement in the metrics. The he_unifrom initializer works well with Relu activation units.

Model 4 Results









Analysis:

- Used he_uniform weights initialisers and relu activations to avoid vanishing gradient,faster convergance and dead activation state problems
- Instead of SGD with momentum, used adam optimizer for auto tuning learning rate.
- Adam adapts the updates to each individual parameters to perform smaller or larger updates depending on their parameter importance. Designed to tackle in large scale problems where its impractical to manually choose different learning rates for each feature due to sheer volume of the dimensions
- Dynamically incorporates knowledge of the geometry of the data observed in past epochs, then applies lower learning rates for dense features and high learning rates for sparse features. Also overcomes saddle points