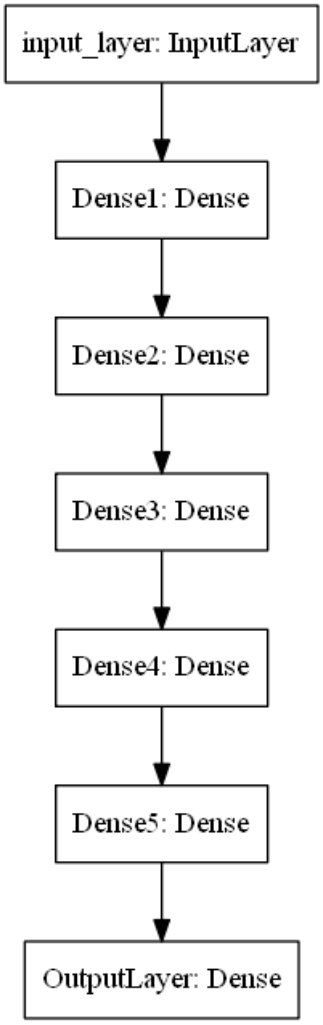
1. Download the data from here

2. Code the model to classify data like below image



3. Write your own callback function, that has to print the micro F1 score and AUC score after each epoch.

4. Save your model at every epoch if your validation accuracy is improved from previous epoch.

5. you have to decay learning based on below conditions
        Cond1. If your validation accuracy at that epoch is less than previous epoch accuracy, you have to decrese the
                learning rate by 10%.
        Cond2. For every 3rd epoch, decay your learning rate by 5%.

6. If you are getting any NaN values(either weigths or loss) while training, you have to terminate your training.

7. You have to stop the training if your validation accuracy is not increased in last 2 epochs.

8. Use tensorboard for every model and analyse your gradients. (you need to upload the screenshots for each model for evaluation)

9. use cross entropy as loss function

10. Try the architecture params as given below.

Model-1
1. Use tanh as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

Model-2
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

Model-3
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use he_uniform() as initilizer.
3. Analyze your output and training process.

Model-4
1. Try with any values to get better accuracy/f1 score.

In [1]:

```
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow.keras.layers import Dense,Activation
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.initializers import RandomUniform
from tensorflow.keras.initializers import HeUniform
from sklearn.model_selection import train_test_split
import datetime, os
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dropout
from sklearn.metrics import f1_score, roc_auc_score
```

In [2]:

```
dataset = pd.read_csv('data.csv')
dataset.head()
```

Out[2]:

	f1	f2	label
0	0.450564	1.074305	0.0
1	0.085632	0.967682	0.0
2	0.117326	0.971521	1.0
3	0.982179	-0.380408	0.0
4	-0.720352	0.955850	0.0

In [3]:

```
X = dataset.copy()
y = X.pop('label')
X = np.array(X)
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size= 0.20)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(16000, 2)
(4000, 2)
(16000,)
(4000,)
```

In [5]:

```
input_dim = X_train.shape[1]
```

```
input_dim
```

Out[5]:

```
2
```

In [6]:

```
from keras import backend as k
# Create custom Callback class
class Callback_Custom(Callback):

    def __init__(self, no, train, val):
        self.no = no
        self.train = train
        self.val = val

    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'acc': [], 'val_acc': [], 'auc': [], 'val_auc': [], 'f1_score': [], 'val_f1_score':

    def on_epoch_end(self, epoch, logs={}):

        # 3. Write your own callback function, that has to print the micro F1 score and AUC score after e
        self.history['acc'].append(logs.get('accuracy'))
        y_train_true = self.train[1]
        y_train_pred = self.model.predict(self.train[0])
        y_train_pred_f1 = y_train_pred.round()
        y_train_pred_f1.astype(int)
        self.history['f1_score'].append(f1_score(y_train_true, y_train_pred_f1, average='micro'))
        self.history['auc'].append(roc_auc_score(y_train_true, y_train_pred))

        if logs.get('val_accuracy', -1) != -1:
            self.history['val_acc'].append(logs.get('val_accuracy'))

        y_val_true = self.val[1]
        y_val_pred = self.model.predict(self.val[0])
        y_val_pred_f1 = y_val_pred.round()
        y_val_pred_f1.astype(int)
        self.history['val_f1_score'].append(f1_score(y_val_true, y_val_pred_f1, average='micro'))
        self.history['val_auc'].append(roc_auc_score(y_val_true, y_val_pred))

        f1 = None
        if (self.history['f1_score'][-1] != None):
            f1 = round(self.history['f1_score'][-1], 4)
        print("Train: accuracy " + str(round(self.history['acc'][-1], 4)) + " auc "
              + str(round(self.history['auc'][-1], 4)) +  " f1-score " + str(f1))

        if (len(self.history['val_acc']) > 0):
            val_f1 = None
            if (self.history['val_f1_score'][-1] != None):
                val_f1 = round(self.history['val_f1_score'][-1], 4)
            print("Val: accuracy " + str(round(self.history['val_acc'][-1], 4)) + " auc "
                + str(round(self.history['val_auc'][-1], 4)) +  " f1-score " + str(val_f1))

        # 4. Save your model at every epoch if your validation accuracy is improved from previous epoch.
        if (len(self.history['val_acc']) >=2 and self.history['val_acc'][-1] != None
            and self.history['val_acc'][-1] > self.history['val_acc'][-2]):
                filepath="model_save/model-" + str(self.no) + "weights-" + str(epoch) + "-" + str(self.his
                self.model.save(filepath)

        #5. Update Learning rate
        lr = k.eval(self.model.optimizer.learning_rate)
        if epoch%3 == 0:
            lr = lr - (lr * 0.05)
            k.set_value(self.model.optimizer.learning_rate, lr)

        if (len(self.history['val_auc']) > 1 and
            self.history['val_auc'][-1] < self.history['val_auc'][-2]):
            lr = k.eval(self.model.optimizer.learning_rate)
            lr = lr - (lr *  0.1)
            k.set_value(self.model.optimizer.learning_rate, lr)

        print("Learning rate:",k.eval(self.model.optimizer.learning_rate))


        # 6. If you are getting any NaN values(either weigths or loss) while training, you have to termi
        loss = logs.get('loss')
```

```
        if loss is not None:
            if np.isnan(loss) or np.isinf(loss):
                print("Invalid loss and terminated at epoch {}".format(epoch+1))
                self.model.stop_training = True
        for w in self.model.get_weights():
             if np.isnan(w).any():
                print("Invalid weight and terminated at epoch {}".format(epoch+1))
                self.model.stop_training = True

        # 7. You have to stop the training if your validation accuracy is not increased in last 2 epochs
        if len(self.history['val_acc']) >= 3:
            if ((self.history['val_acc'][-1] < self.history['val_acc'][-3]) and
               (self.history['val_acc'][-1] < self.history['val_acc'][-2])):
                    print("Validation accuracy not improving and terminated at epoch {}".format(epoch+1))
                    self.model.stop_training = True
```

In [7]:

```
%load_ext tensorboard
```

In [8]:

```
def createModel(active, opti, initlizer):
    model = Sequential()
    model.add(Dense(2,activation=active,kernel_initializer=initlizer,
                    kernel_regularizer=tf.keras.regularizers.l1(0.01), input_dim=input_dim))
    model.add(Dense(2,activation=active,kernel_initializer=initlizer,
                    kernel_regularizer=tf.keras.regularizers.l1(0.01)))
    model.add(Dense(2,activation=active,kernel_initializer=initlizer,
                    kernel_regularizer=tf.keras.regularizers.l1(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(2,activation=active,kernel_initializer=initlizer,
                    kernel_regularizer=tf.keras.regularizers.l1(0.01)))
    model.add(Dense(2,activation=active,kernel_initializer=initlizer,
                    kernel_regularizer=tf.keras.regularizers.l1(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(1,activation='sigmoid',kernel_initializer=initlizer,
                    kernel_regularizer=tf.keras.regularizers.l1(0.01)))

    model.compile(optimizer=opti,loss='binary_crossentropy',metrics=['accuracy'])
    return model
```

Model 1

In [9]:

```
# Model 1
logdir1 = os.path.join("logs_model1", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback1 = tf.keras.callbacks.TensorBoard(log_dir=logdir1,histogram_freq=1, write_graph=True

callback_custom1 = Callback_Custom(1, (X_train, y_train), (X_test, y_test))
optimizer1 = tf.keras.optimizers.SGD(learning_rate = 0.001, momentum = 0.99)
initializer1 = RandomUniform(minval=0, maxval=1)
model1 = createModel('tanh', optimizer1, initializer1)
model1.fit(X_train,y_train,epochs=20, validation_data=(X_test,y_test),
           batch_size=20, callbacks=[callback_custom1, tensorboard_callback1])
```

```
Epoch 1/20
  3/800 [..............................] - ETA: 3:07 - loss: 1.1370 - accuracy:
0.4306WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch
time: 0.0031s vs `on_train_batch_end` time: 0.0809s). Check your callbacks.
800/800 [==============================] - 3s 3ms/step - loss: 0.8135 - accuracy: 0.4963 - val_loss: 0.6
981 - val_accuracy: 0.5000
Train: accuracy 0.4985 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00095
Epoch 2/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6951 - accuracy: 0.5032 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.5039 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00095
Epoch 3/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6936 - accuracy: 0.4978 - val_loss: 0.6
935 - val_accuracy: 0.5000
Train: accuracy 0.497 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00095
Epoch 4/20
```

800/800 [==============================] - 1s 2ms/step - loss: 0.6936 - accuracy: 0.4926 - val_loss: 0.6934 - val_accuracy: 0.5000
Train: accuracy 0.4966 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0009025
Epoch 5/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6934 - accuracy: 0.5047 - val_loss: 0.6935 - val_accuracy: 0.5000
Train: accuracy 0.504 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0009025
Epoch 6/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6935 - accuracy: 0.5005 - val_loss: 0.6942 - val_accuracy: 0.5000
Train: accuracy 0.5033 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0009025
Epoch 7/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6934 - accuracy: 0.5012 - val_loss: 0.6932 - val_accuracy: 0.5000
Train: accuracy 0.4991 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.000857375
Epoch 8/20
800/800 [==============================] - 2s 3ms/step - loss: 0.6933 - accuracy: 0.5056 - val_loss: 0.6932 - val_accuracy: 0.5000
Train: accuracy 0.5018 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.000857375
Epoch 9/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6933 - accuracy: 0.5060 - val_loss: 0.6932 - val_accuracy: 0.5000
Train: accuracy 0.5002 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.000857375
Epoch 10/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6937 - accuracy: 0.4959 - val_loss: 0.6935 - val_accuracy: 0.5000
Train: accuracy 0.4961 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0008145062
Epoch 11/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6937 - accuracy: 0.4987 - val_loss: 0.6932 - val_accuracy: 0.5000
Train: accuracy 0.5013 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0008145062
Epoch 12/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6933 - accuracy: 0.4921 - val_loss: 0.6934 - val_accuracy: 0.5000
Train: accuracy 0.4975 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0008145062
Epoch 13/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6934 - accuracy: 0.4989 - val_loss: 0.6932 - val_accuracy: 0.5000
Train: accuracy 0.4963 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00077378086
Epoch 14/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6934 - accuracy: 0.4956 - val_loss: 0.6932 - val_accuracy: 0.5000
Train: accuracy 0.4949 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00077378086
Epoch 15/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6933 - accuracy: 0.5026 - val_loss: 0.6937 - val_accuracy: 0.5000
Train: accuracy 0.5011 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00077378086
Epoch 16/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6936 - accuracy: 0.4935 - val_loss: 0.6933 - val_accuracy: 0.5000
Train: accuracy 0.4934 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0007350918

```
Epoch 17/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6934 - accuracy: 0.5028 - val_loss: 0.6
939 - val_accuracy: 0.5000
Train: accuracy 0.5038 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0007350918
Epoch 18/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6935 - accuracy: 0.5032 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4989 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0007350918
Epoch 19/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6934 - accuracy: 0.5039 - val_loss: 0.6
933 - val_accuracy: 0.5000
Train: accuracy 0.499 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006983372
Epoch 20/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6935 - accuracy: 0.4983 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4974 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006983372
```

Out[9]:

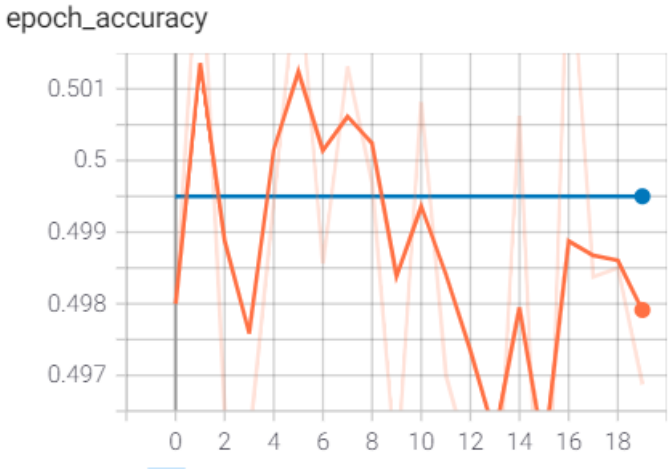
```
<tensorflow.python.keras.callbacks.History at 0x21521cbc5f8>
```

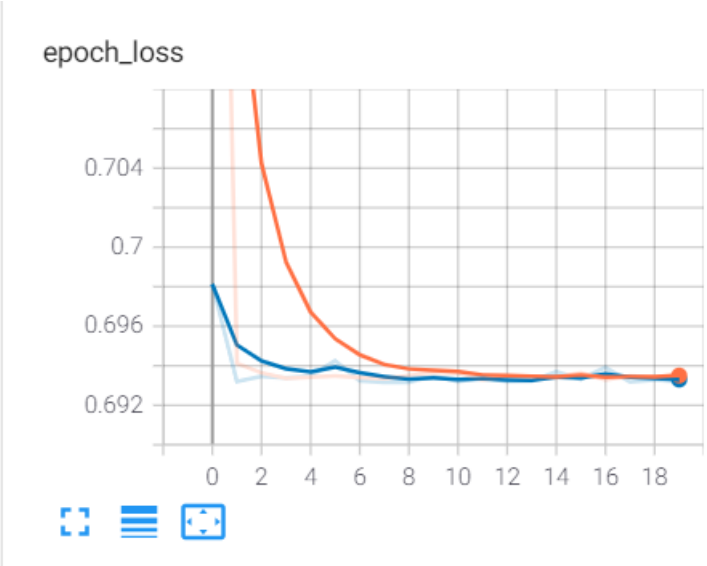
In [10]:

```
%tensorboard --logdir logs_model1
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Model 1 accuracy and loss

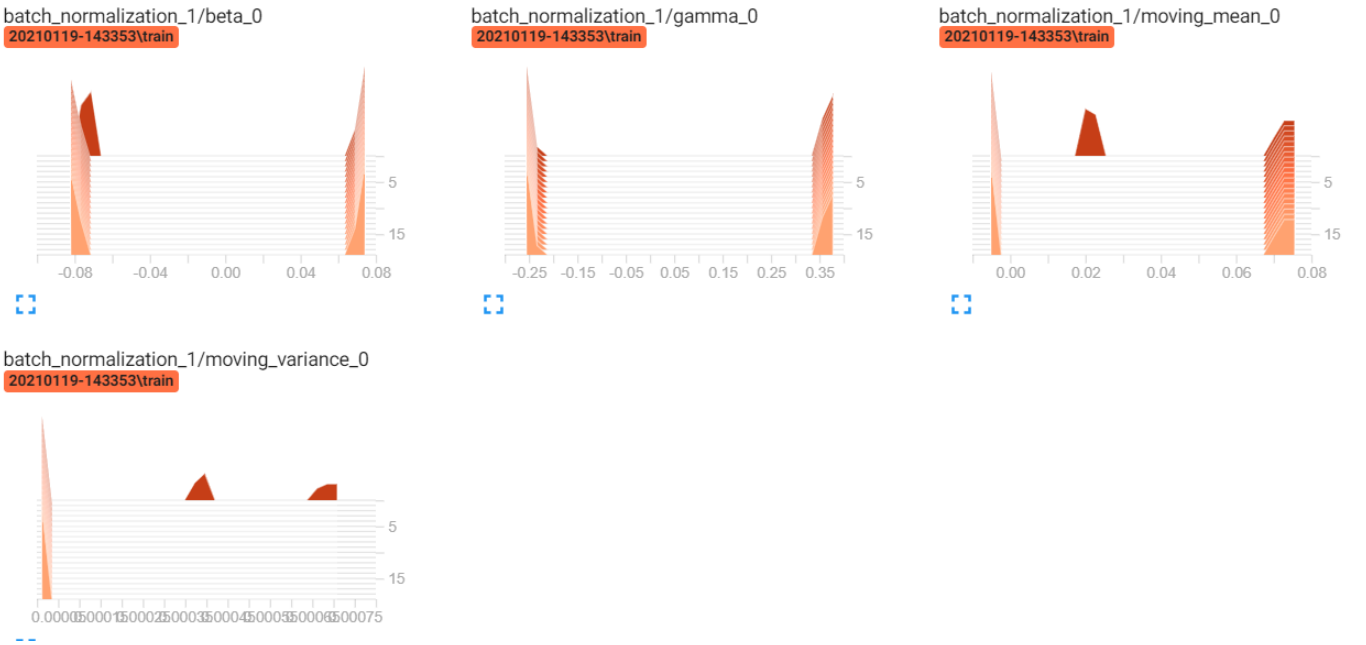




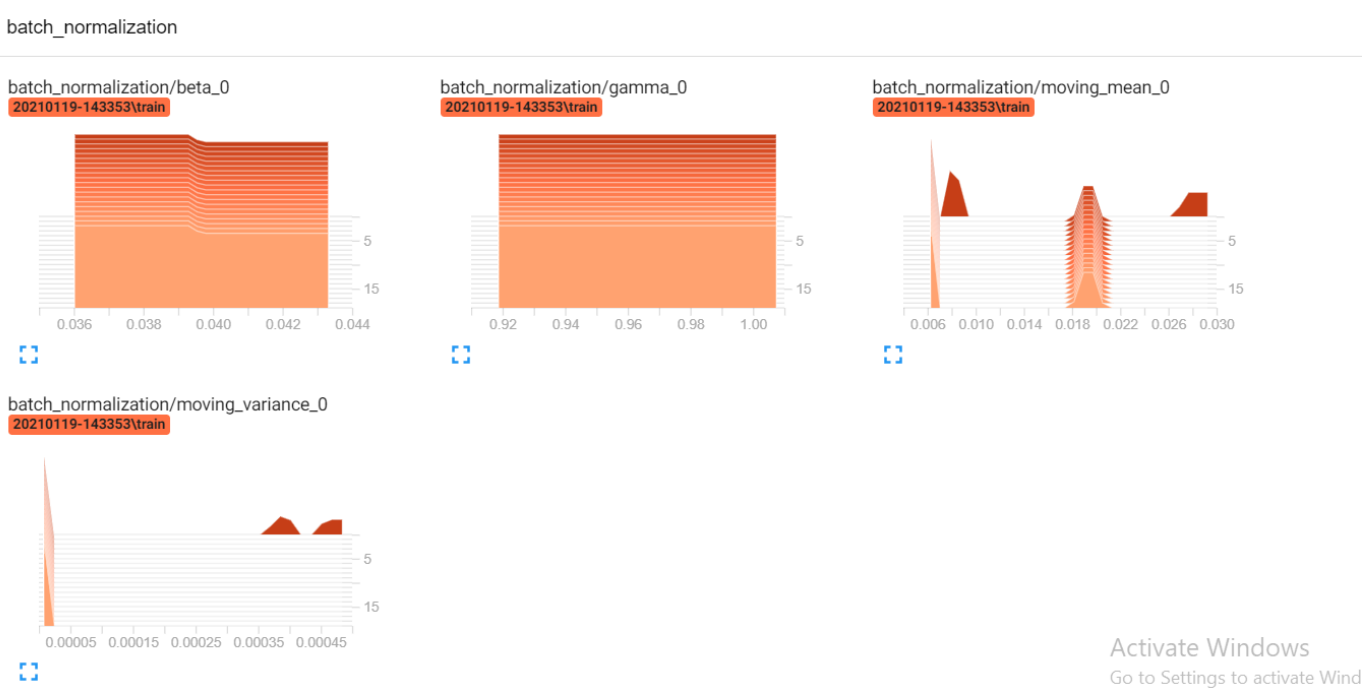
In model 1, we got epoch accuracy as 0.5 in all epochs and its more than trainning accuracy. So SGD with momentum is giving good result.

The training and validation loss are converging as epoch number increases.

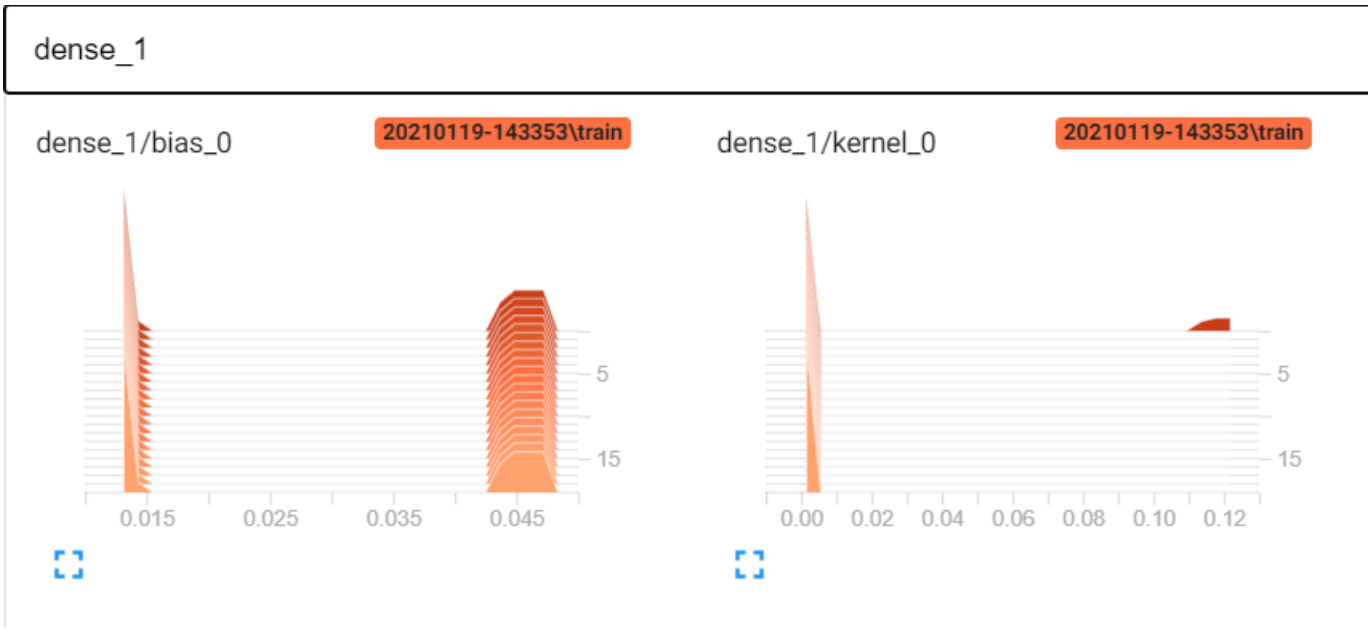
Distribution of mean, variance, scale and shift in Batch Normalization 1



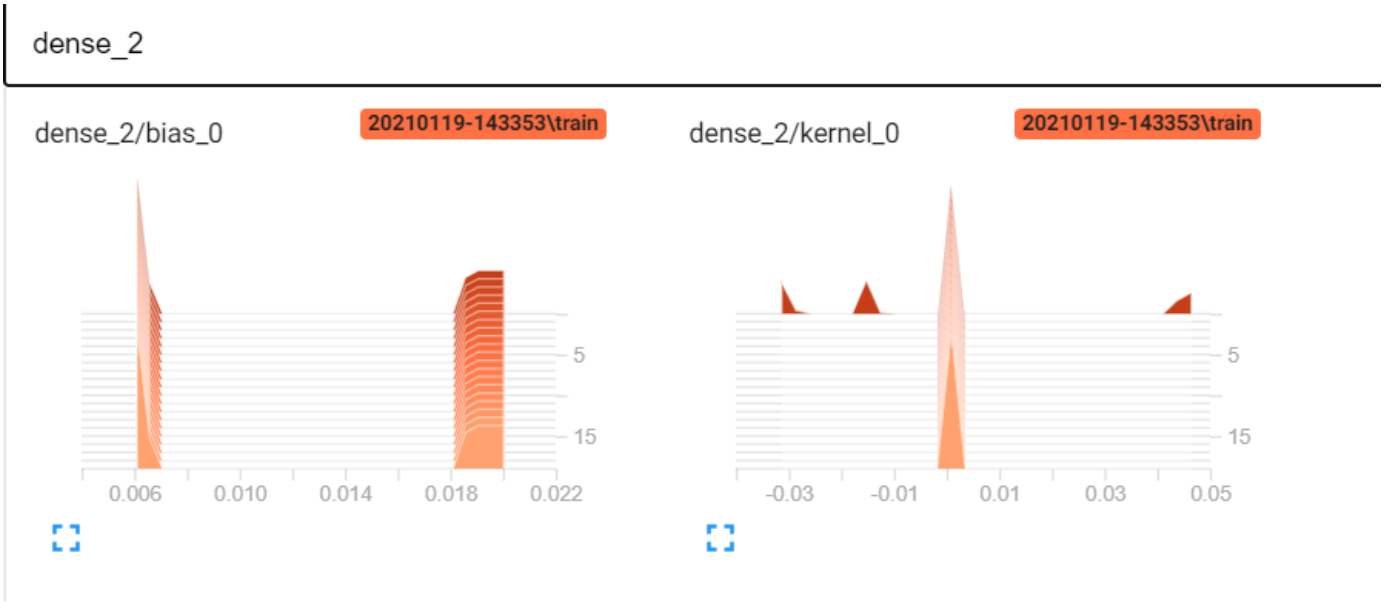
Distribution of mean, variance, scale and shift in Batch Normalization 2



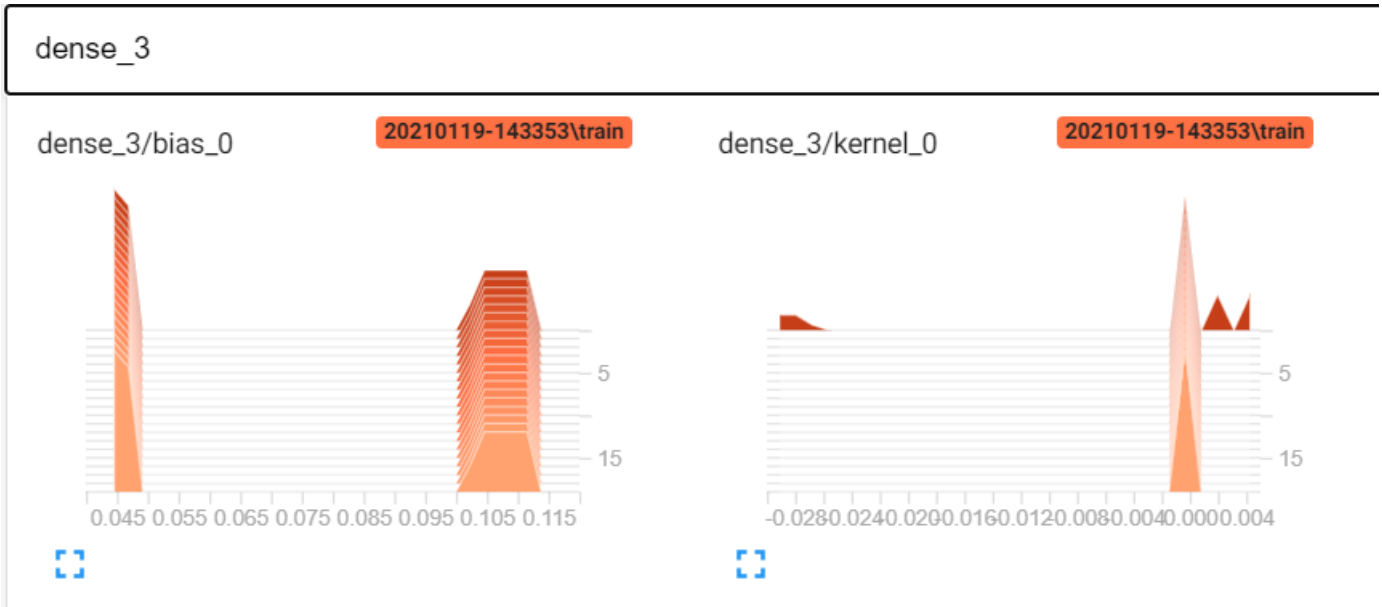
Distribution of weight and bias for layer 1



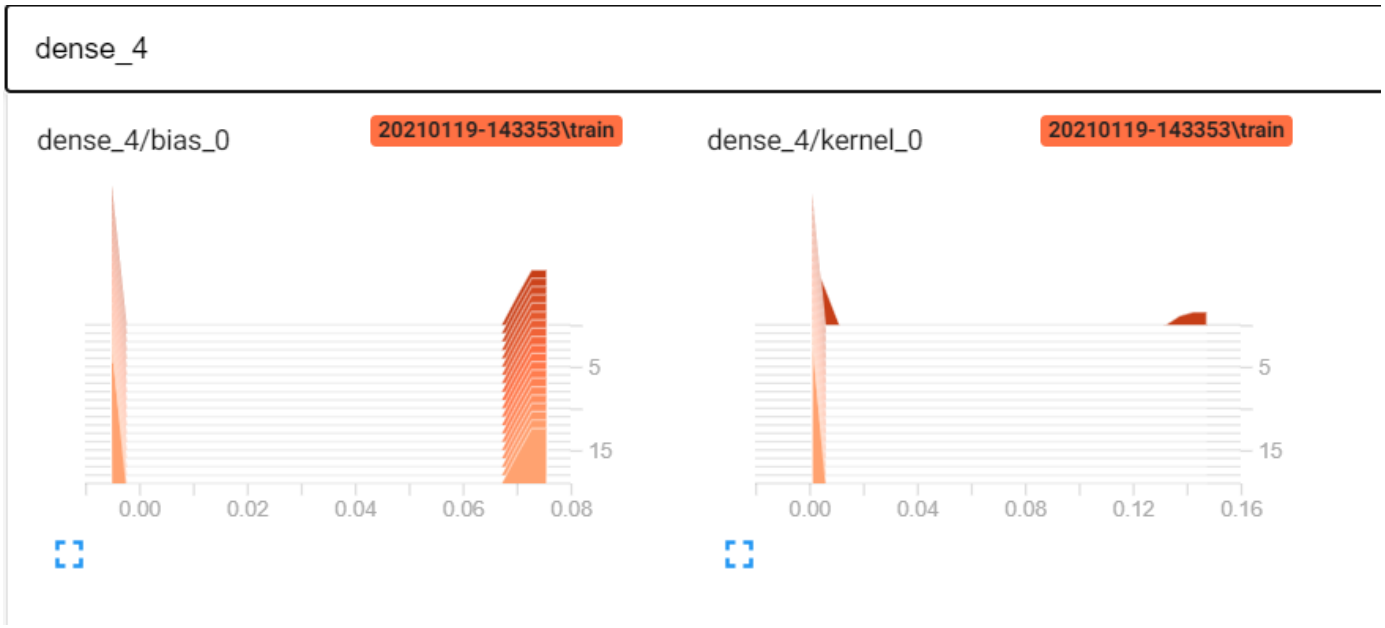
Distribution of weight and bias for layer 2



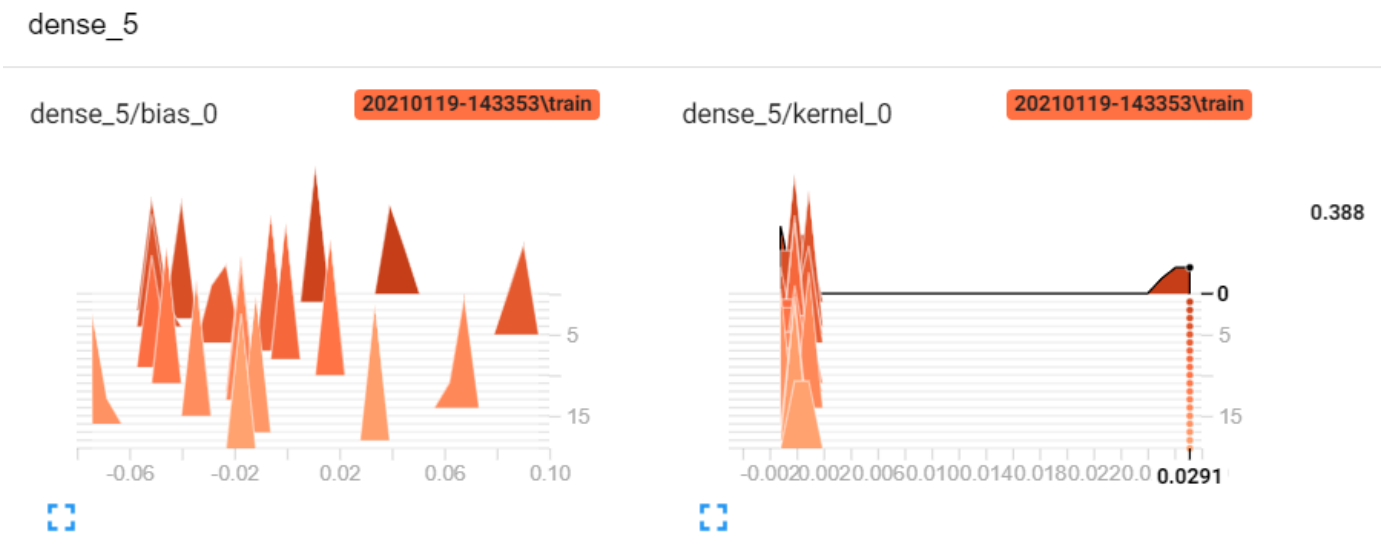
Distribution of weight and bias for layer 3



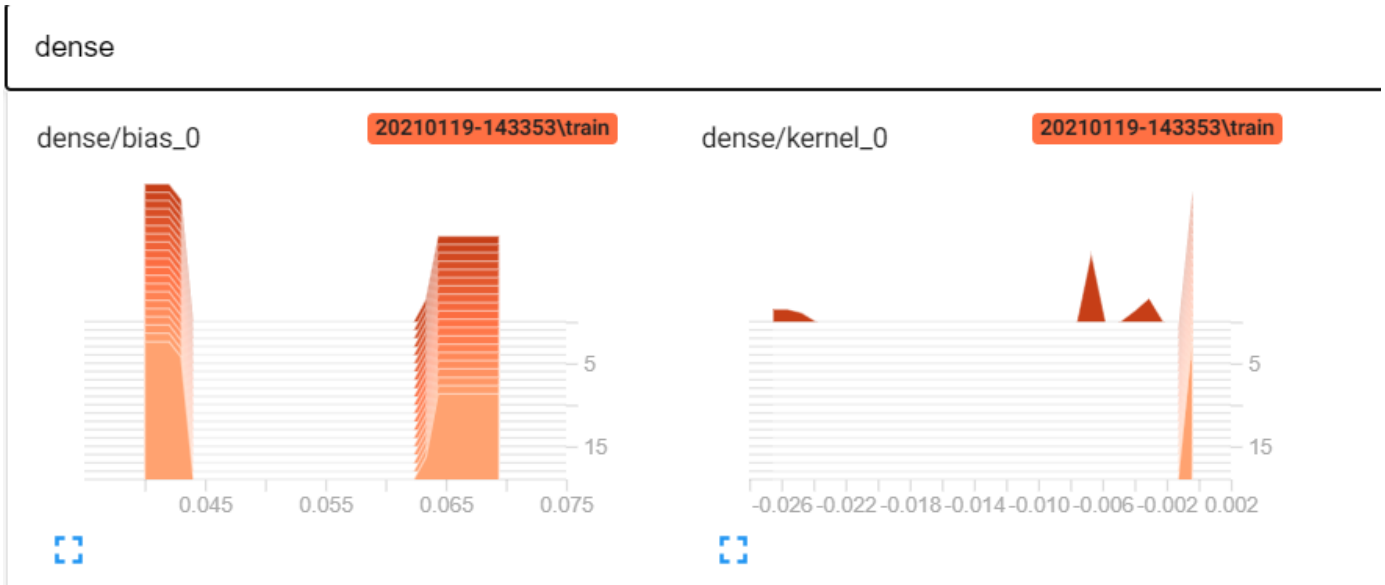
Distribution of weight and bias for layer 4



Distribution of weight and bias for layer 5



Distribution of weight and bias for output layer



In the hidden layer, the range of values are very small, but we have few outliers but values are within 0,1.

In [10]:

```
# Model 2
logdir2 = os.path.join("logs_model2", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback2 = tf.keras.callbacks.TensorBoard(log_dir=logdir2,histogram_freq=1, write_graph=True
```

```
callback_custom2 = Callback_Custom(2, (X_train, y_train), (X_test, y_test))
optimizer2 = tf.keras.optimizers.SGD(learning_rate = 0.001, momentum = 0.95)
initializer2 = RandomUniform(minval=0, maxval=1)
model2 = createModel('relu', optimizer2, initializer2)
model2.fit(X_train,y_train,epochs=20, validation_data=(X_test,y_test),
           batch_size=20, callbacks=[callback_custom2, tensorboard_callback2])
```

```
Epoch 1/20
  3/800 [..............................] - ETA: 3:48 - loss: 1.1278 - accuracy:
0.3750WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch
time: 0.0060s vs `on_train_batch_end` time: 0.0936s). Check your callbacks.
800/800 [==============================] - 3s 3ms/step - loss: 0.8292 - accuracy: 0.4964 - val_loss: 0.7
629 - val_accuracy: 0.5278
Train: accuracy 0.507 auc 0.5427 f1-score 0.5343
Val: accuracy 0.5278 auc 0.539 f1-score 0.5278
Learning rate: 0.00095
Epoch 2/20
800/800 [==============================] - 1s 1ms/step - loss: 0.7570 - accuracy: 0.5175 - val_loss: 0.7
398 - val_accuracy: 0.5520
Train: accuracy 0.5203 auc 0.5554 f1-score 0.5477
Val: accuracy 0.552 auc 0.5519 f1-score 0.552
Learning rate: 0.00095
Epoch 3/20
800/800 [==============================] - 1s 1ms/step - loss: 0.7338 - accuracy: 0.5141 - val_loss: 0.7
225 - val_accuracy: 0.5470
Train: accuracy 0.5141 auc 0.5457 f1-score 0.5511
Val: accuracy 0.547 auc 0.5413 f1-score 0.547
Learning rate: 0.000855
Epoch 4/20
800/800 [==============================] - 1s 2ms/step - loss: 0.7197 - accuracy: 0.5004 - val_loss: 0.7
115 - val_accuracy: 0.5000
Train: accuracy 0.5063 auc 0.5297 f1-score 0.5
Val: accuracy 0.5 auc 0.527 f1-score 0.5
Learning rate: 0.000731025
Validation accuracy not improving and terminated at epoch 4
```

Out[10]:

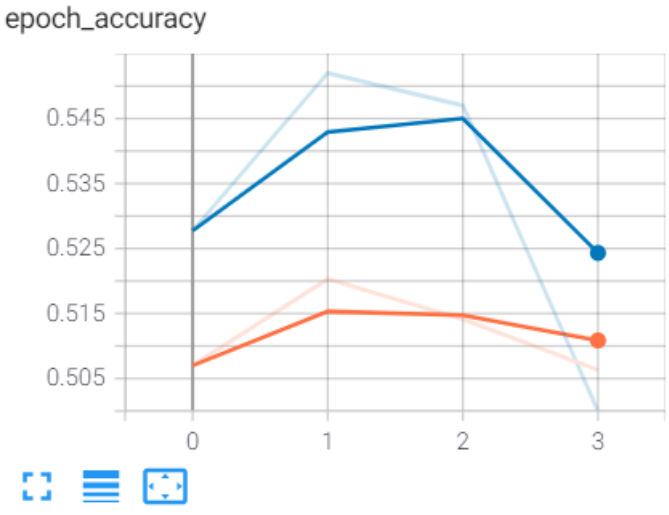
```
<tensorflow.python.keras.callbacks.History at 0x215224164e0>
```

In [12]:

```
%tensorboard --logdir logs_model2
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Accuracy and loss for model 2



Here the validation accuracy is more than the training and also the loss is less than the training. But as the accuracy didnt improve for past 2 epochs, we had to stop training.

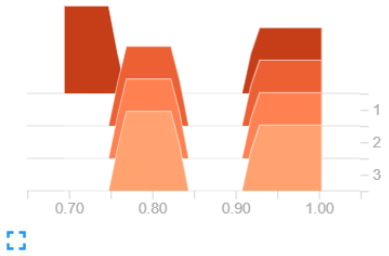
Distribution of mean, variance, scale, shift in Batch Normalization 1

Distribution of mean, variance, scale, shift in Batch Normalization 2

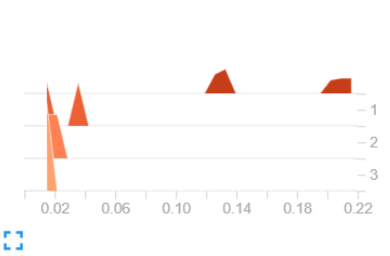
batch_normalization_3

batch_normalization_3/beta_0
20210119-143541\train

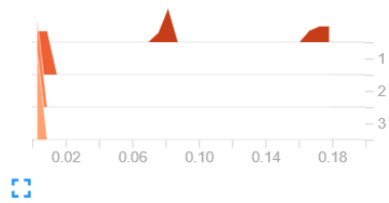
batch_normalization_3/gamma_0
20210119-143541\train



batch_normalization_3/moving_mean_0
20210119-143541\train



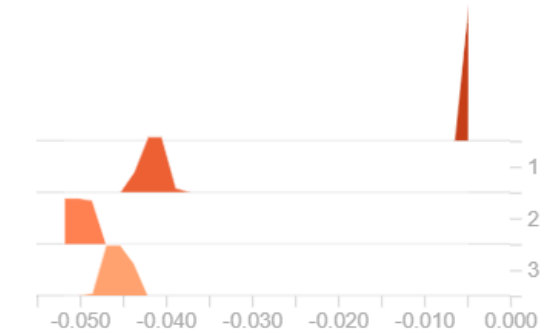
batch_normalization_3/moving_variance_0
20210119-143541\train



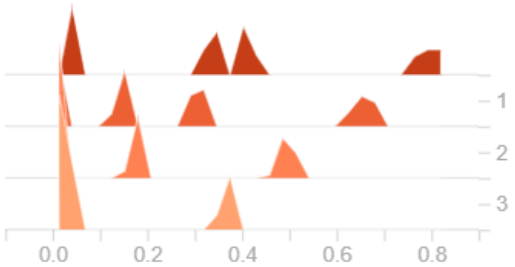
Weight and bias distribution layer 1

dense_6

dense_6/bias_0
20210119-143541\train



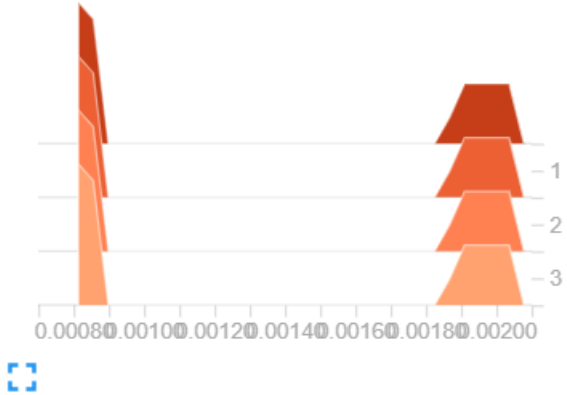
dense_6/kernel_0
20210119-143541\train



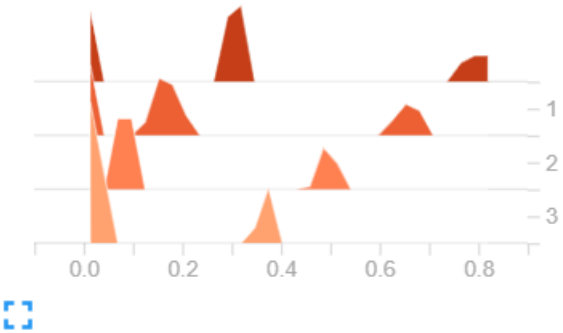
Weight and bias distribution layer 2

dense_7

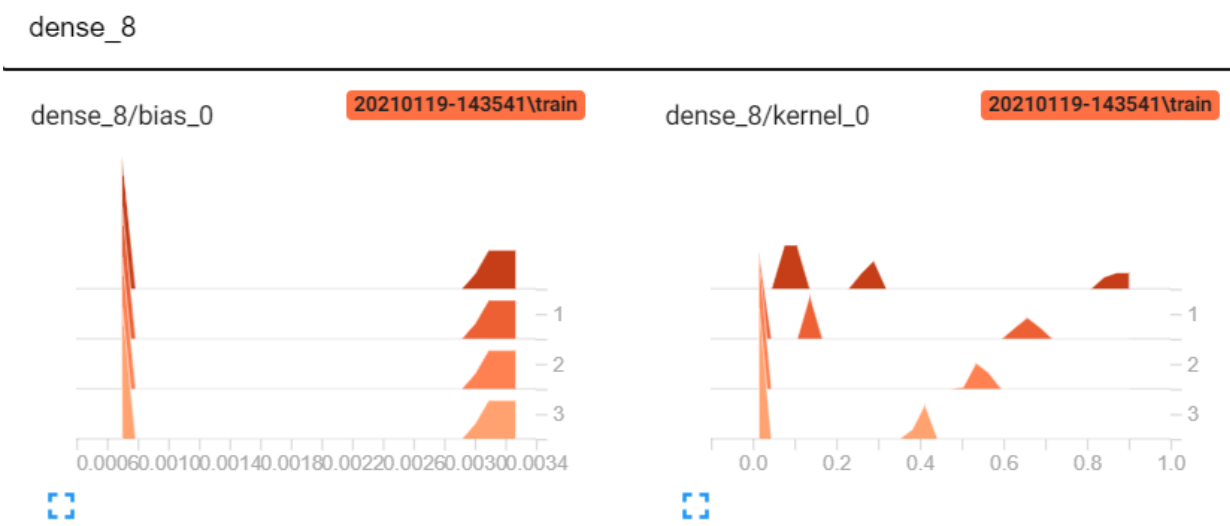
dense_7/bias_0
20210119-143541\train



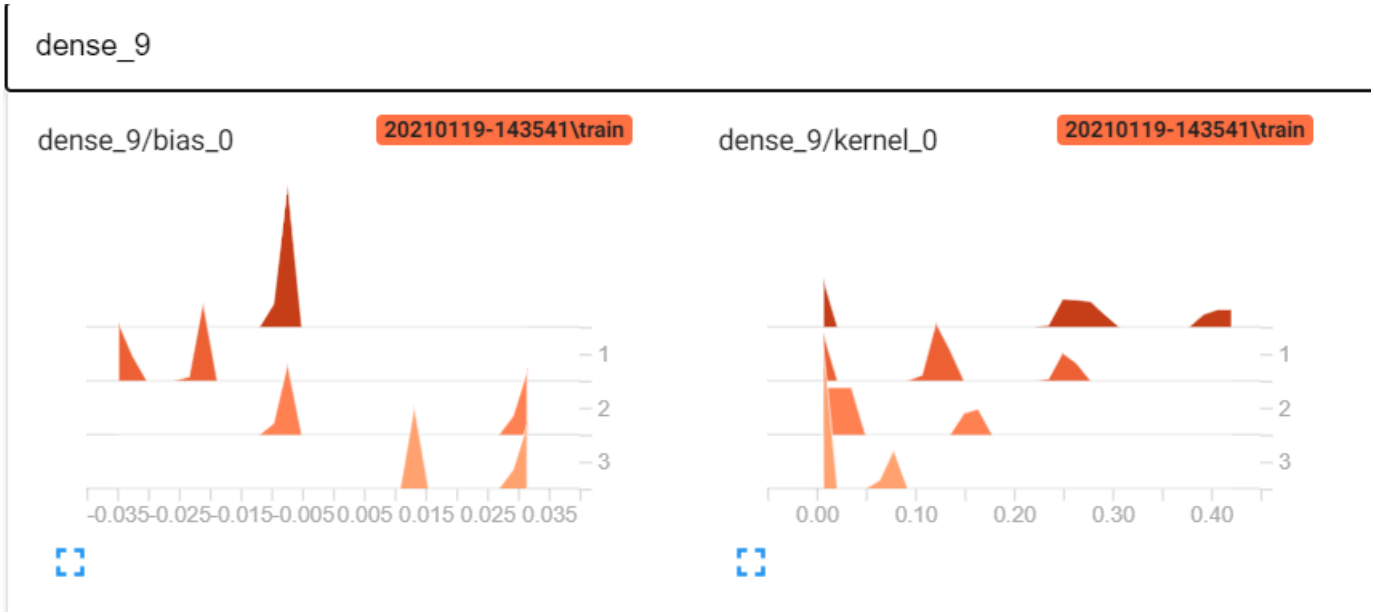
dense_7/kernel_0
20210119-143541\train



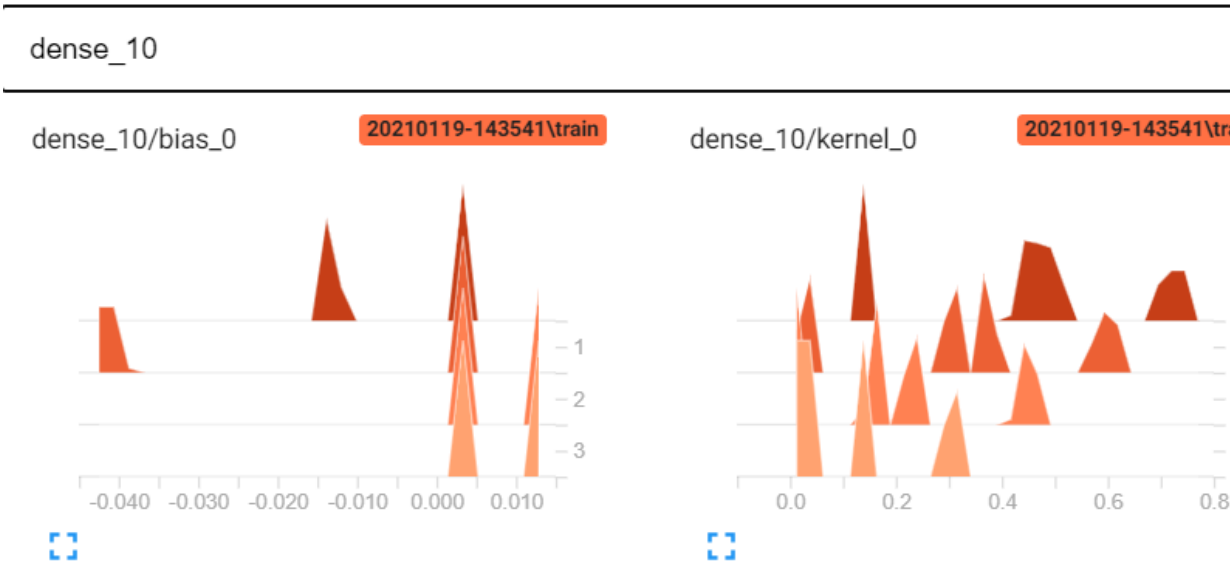
Weight and bais distribution layer 3



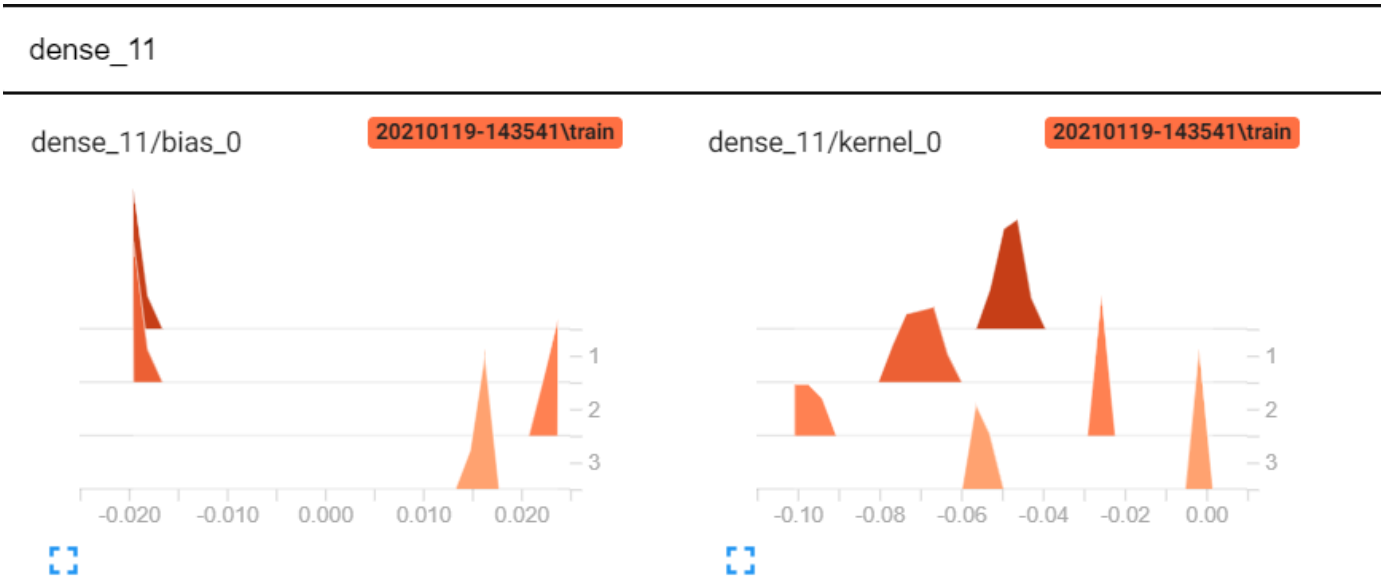
Weight and bias distribution layer 4



Weight and bais distribution layer 5



Weight and bias distribution output layer



As we didnt have more epochs, we didnt get proper weight and bias.

In [11]:

```
# Model 3
logdir3 = os.path.join("logs_model3", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback3 = tf.keras.callbacks.TensorBoard(log_dir=logdir3,histogram_freq=1, write_graph=True


callback_custom3 = Callback_Custom(3, (X_train, y_train), (X_test, y_test))
optimizer3 = tf.keras.optimizers.SGD(learning_rate = 0.001, momentum = 0.95)
initializer3 = HeUniform(seed=None)
model3 = createModel('relu', optimizer3, initializer3)
model2.fit(X_train,y_train,epochs=20, validation_data=(X_test,y_test),
           batch_size=20, callbacks=[callback_custom3, tensorboard_callback3])
```

```
Epoch 1/20
  3/800 [..............................] - ETA: 4:14 - loss: 0.7107 - accuracy:
0.5500WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch
time: 0.0051s vs `on_train_batch_end` time: 0.1089s). Check your callbacks.
800/800 [==============================] - 2s 3ms/step - loss: 0.7069 - accuracy: 0.4951 - val_loss: 0.7
034 - val_accuracy: 0.5000
Train: accuracy 0.4951 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00069447374
Epoch 2/20
800/800 [==============================] - 2s 2ms/step - loss: 0.7010 - accuracy: 0.4978 - val_loss: 0.6
987 - val_accuracy: 0.5000
Train: accuracy 0.4978 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00069447374
Epoch 3/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6966 - accuracy: 0.5002 - val_loss: 0.6
947 - val_accuracy: 0.5000
Train: accuracy 0.5002 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00069447374
Epoch 4/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6936 - accuracy: 0.5038 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.5038 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006597501
Epoch 5/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4969 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4969 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006597501
Epoch 6/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4944 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4944 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006597501
Epoch 7/20
```

800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4941 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4941 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006267626
Epoch 8/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6932 - accuracy: 0.4996 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4996 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006267626
Epoch 9/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4964 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4964 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0006267626
Epoch 10/20
800/800 [==============================] - 2s 3ms/step - loss: 0.6932 - accuracy: 0.4956 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4956 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00059542444
Epoch 11/20
800/800 [==============================] - 1s 1ms/step - loss: 0.6932 - accuracy: 0.4926 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4926 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00059542444
Epoch 12/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4961 - val_loss: 0.6
931 - val_accuracy: 0.5000
Train: accuracy 0.4961 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00059542444
Epoch 13/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6932 - accuracy: 0.4933 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4933 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0005656532
Epoch 14/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6932 - accuracy: 0.4978 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4978 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0005656532
Epoch 15/20
800/800 [==============================] - 2s 3ms/step - loss: 0.6932 - accuracy: 0.4976 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4976 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.0005656532
Epoch 16/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4989 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4989 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00053737056
Epoch 17/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4985 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.4985 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00053737056
Epoch 18/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.5001 - val_loss: 0.6
932 - val_accuracy: 0.5000
Train: accuracy 0.5001 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.00053737056
Epoch 19/20
800/800 [==============================] - 2s 2ms/step - loss: 0.6932 - accuracy: 0.4934 - val_loss: 0.6
931 - val_accuracy: 0.5000
Train: accuracy 0.4934 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.000510502

```
Epoch 20/20
800/800 [==============================] - 1s 2ms/step - loss: 0.6932 - accuracy: 0.4994 - val_loss: 0.6
931 - val_accuracy: 0.5000
Train: accuracy 0.4994 auc 0.5 f1-score 0.5
Val: accuracy 0.5 auc 0.5 f1-score 0.5
Learning rate: 0.000510502
```

Out[11]:

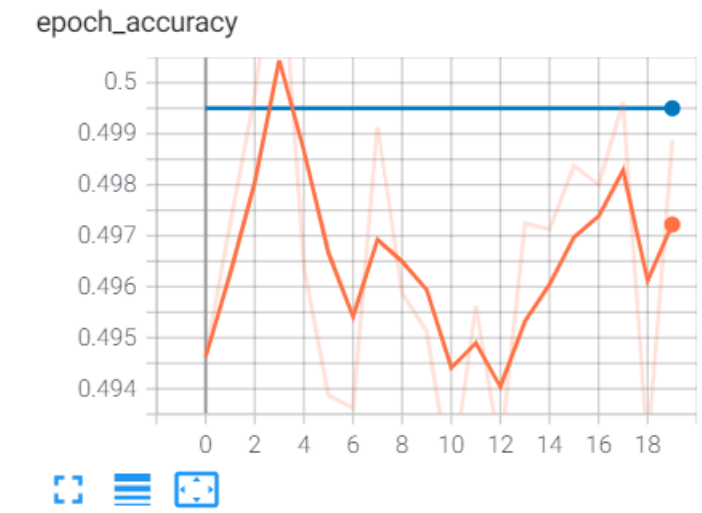
```
<tensorflow.python.keras.callbacks.History at 0x215225bb5c0>
```

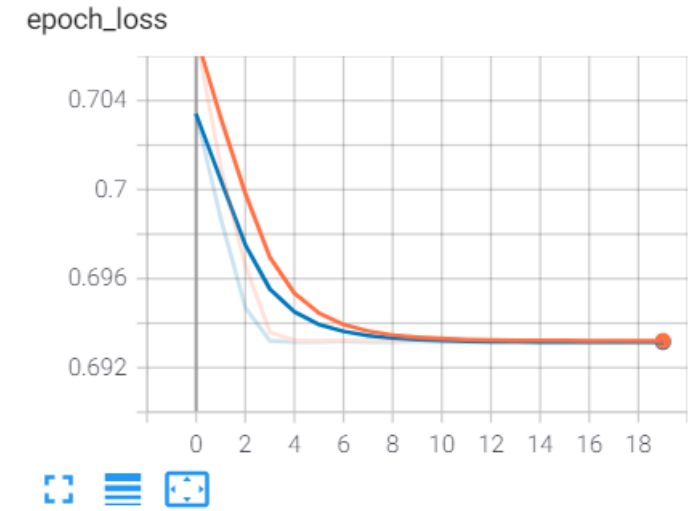
In [14]:

```
%tensorboard --logdir logs_model3
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Accuracy and loss model 3

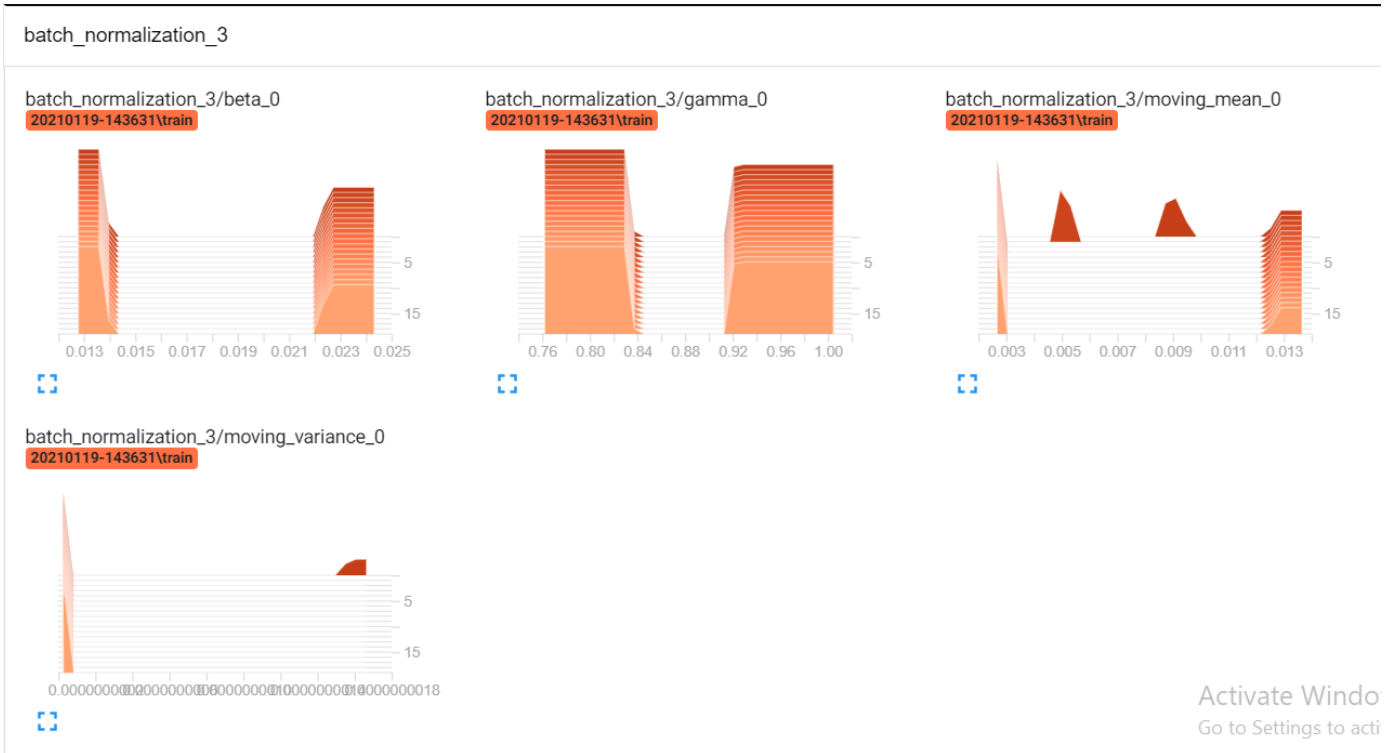




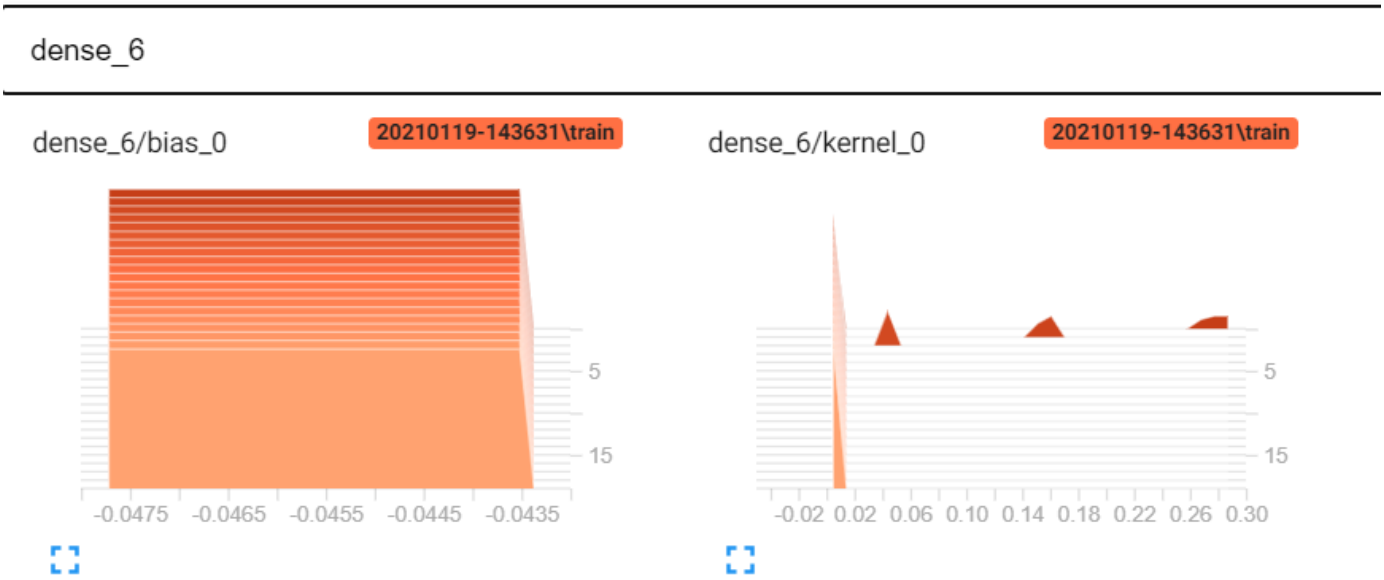
Here also the validation accuracy is more than the training and loss is similar to training for more epochs.

Mean, variance, scale and shift distribution in Batch normalization 1

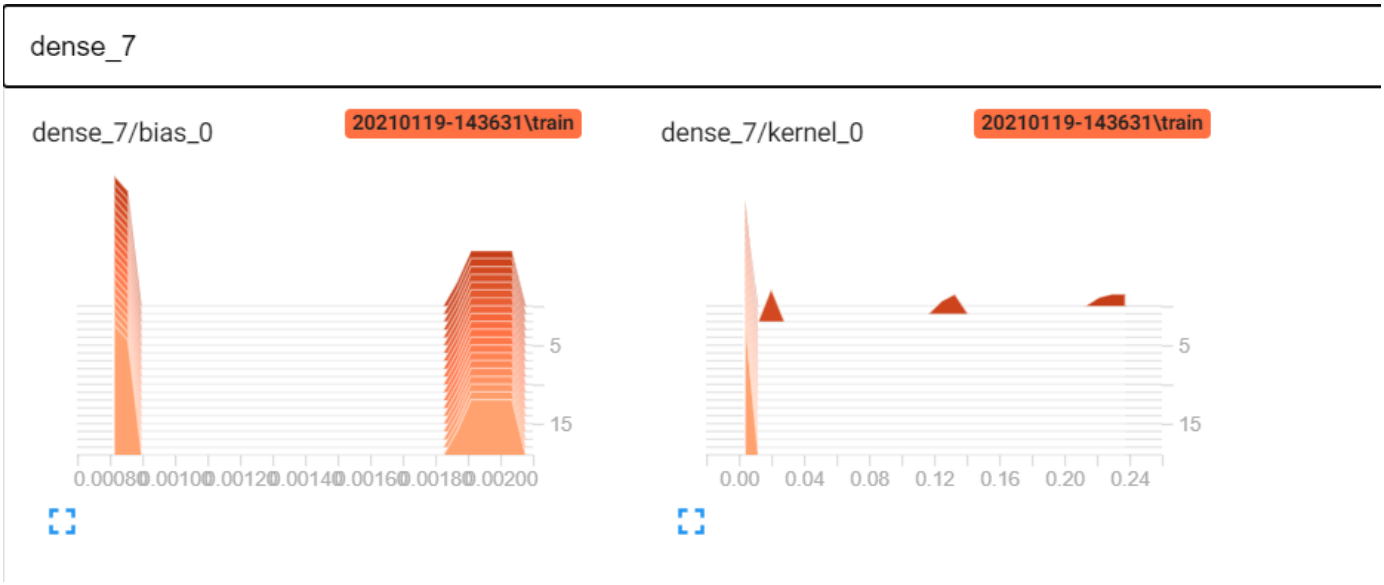
Mean, variance, scale and shift distribution in Batch normalization 2



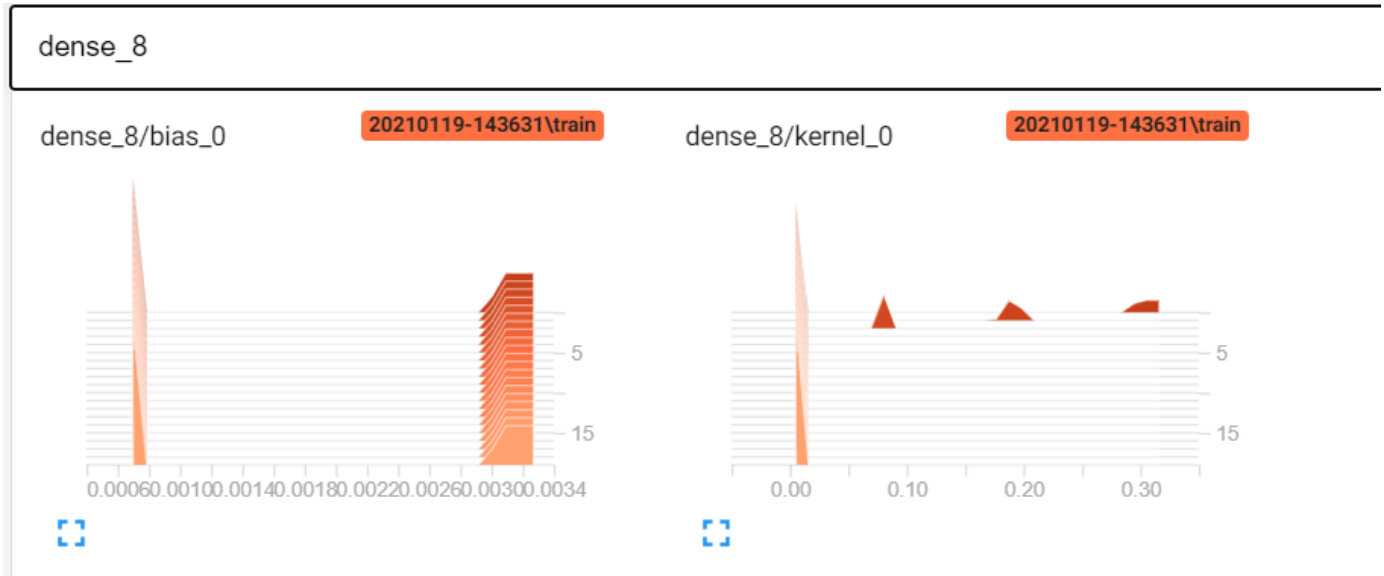
Weight and bais distribution layer 1



Weight and bias distribution layer 2

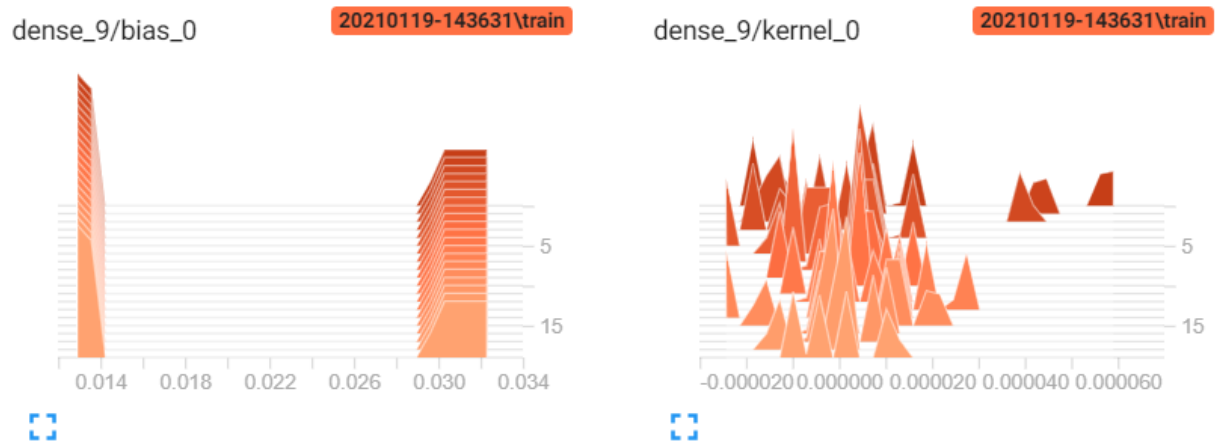


Weight and bias distribution layer 3



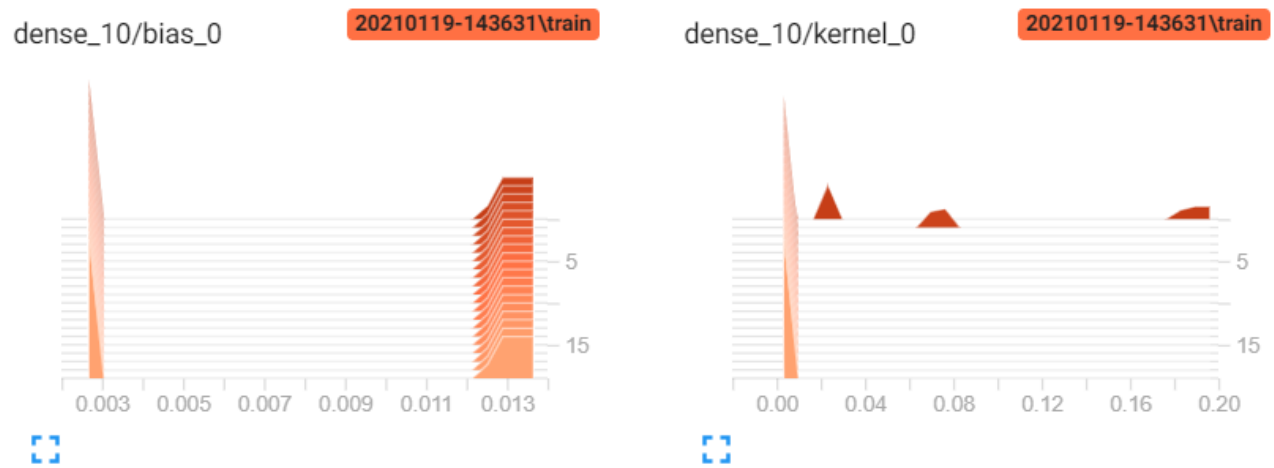
Weight and bias distribution layer 4

dense_9



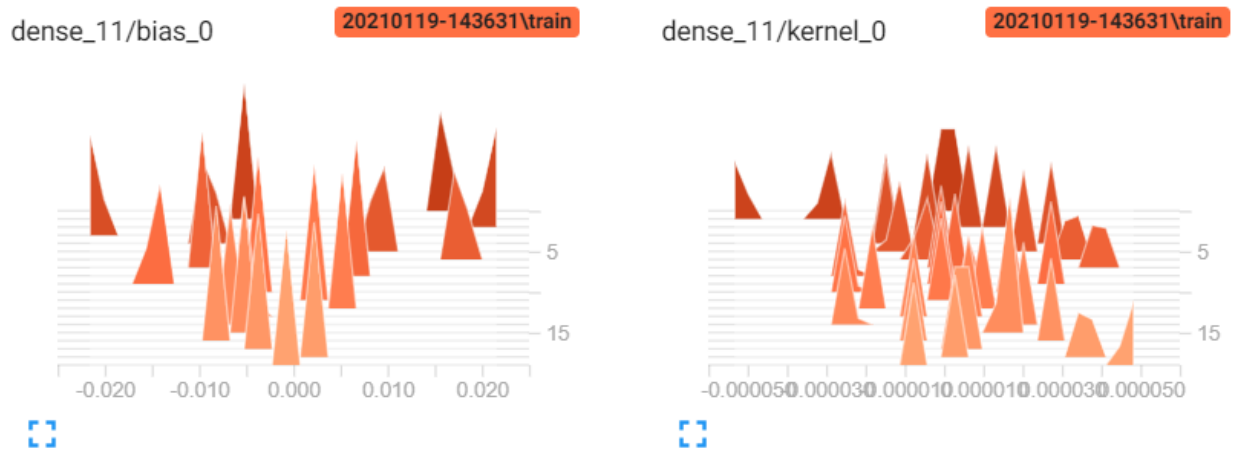
Weight and bais distribution layer 5

dense_10



Weight and bias distribution output layer

dense_11



The distribution of weights are according to He-distribution.

Here for sigmoid and relu activations, we got similar loss. If we had complex data and also more deep layers, relu could have been more useful.