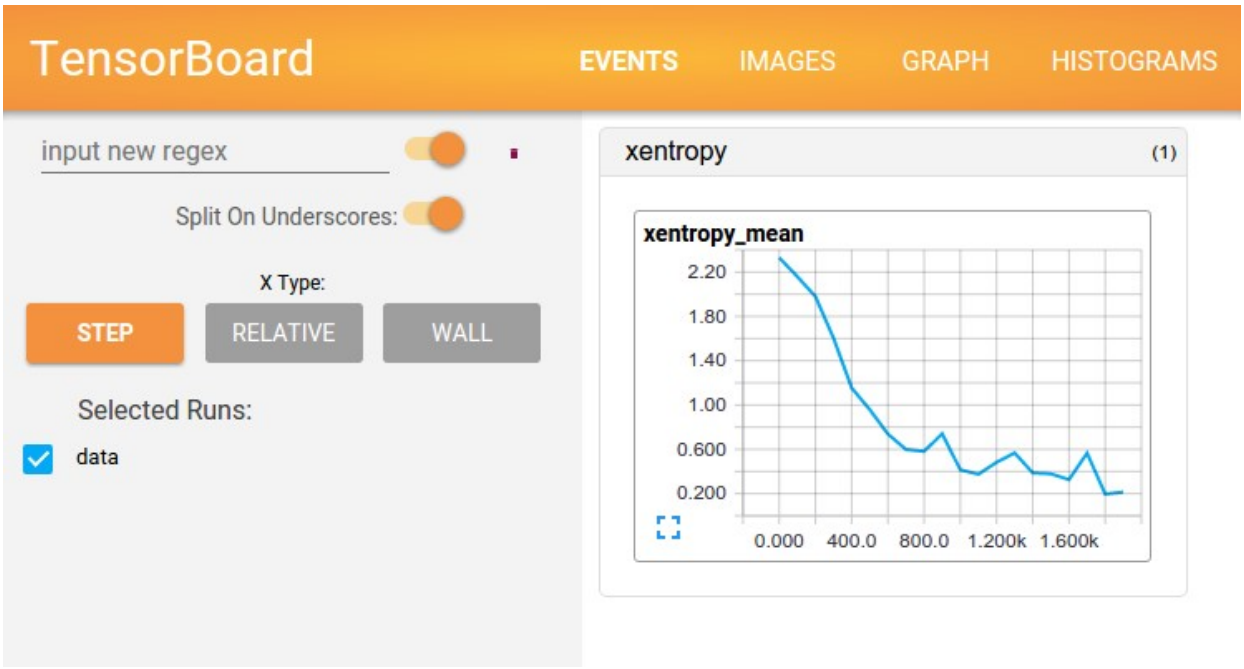
# TensorBoard: Visualizing Learning

## TensorBoard: Visualizing Learning

- TensorFlow을 가지고 아주 큰 deep neural network를 학습시키는 것은 복잡하고 혼란스러울 수 있음.
- DNN을 쉽게 이해하고 디버깅, 최적화하기 위해서 TensorBoard라는 시각화 도구를 포함시킴.
- TensorBoard는 TensorFlow graph를 시각화하고 학습중인 메트릭스를 그려주고 추가적인 데이터를 보여줌.




## Serializing the data

- TensorBoard는 TensorFlow가 동작하면서 생성되는 TensorFlow events files을 읽어오는 것으로 동작함.
- tf$summary$ scalar 에 노드들의 결과물인 learning rate 과 loss 을 저장할 수 있으며,'learning rate' or 'loss function'과 같이 태그를 달아서 구분할 수 있음.
- tf$summary$ histogram 에는 gradient outputs 과 weights 값을 저장하여 각각의 Layer에서의 값들의 분포를 시각화함.
- tf$summary$ merge_all 는 모든 summary data을 한곳으로 결합시키는 명령어
- tf$summary$ FileWriter는 summary data을 디스크에 저장하는 명령어
    - 아래 예제에서는 /tmp/mnist_logs 디렉토리에 summary data을 저장하도록 되어 있음.

```
library(tensorflow)

flags <- tf$app$flags
flags$DEFINE_boolean('fake_data', FALSE, 'If true, uses fake data for unit t
esting.')
flags$DEFINE_integer('max_steps', 1000L, 'Number of steps to run trainer.')
flags$DEFINE_float('learning_rate', 0.001, 'Initial learning rate.')
flags$DEFINE_float('dropout', 0.9, 'Keep probability for training dropout.')
flags$DEFINE_string('summaries_dir', '/tmp/mnist_logs', 'Summaries director
y')
FLAGS <- parse_flags()

train <- function() {
  # Import data
  datasets <- tf$contrib$learn$datasets
  mnist <- datasets$mnist$read_data_sets("MNIST-data", one_hot = TRUE)

  sess <- tf$InteractiveSession()

  # Create a multilayer model.

  # Input placeholders
  with(tf$name_scope("input"), {
    x <- tf$placeholder(tf$float32, shape(NULL, 784L), name = "x-input")
    y_ <- tf$placeholder(tf$float32, shape(NULL, 10L), name = "y-input")
  })

  with(tf$name_scope("input_reshape"), {
    image_shaped_input <- tf$reshape(x, c(-1L, 28L, 28L, 1L))
    tf$summary$image("input", image_shaped_input, 10L)
  })

  # We can't initialize these variables to 0 - the network will get stuck.
  weight_variable <- function(shape) {
    initial <- tf$truncated_normal(shape, stddev = 0.1)
    tf$Variable(initial)
  }

  bias_variable <- function(shape) {
    initial <- tf$constant(0.1, shape = shape)
    tf$Variable(initial)
  }

  # Attach a lot of summaries to a Tensor
  variable_summaries <- function(var, name) {
    with(tf$name_scope("summaries"), {
      mean <- tf$reduce_mean(var)
      tf$summary$scalar(paste0("mean/", name), mean)
      with(tf$name_scope("stddev"), {
        stddev <- tf$sqrt(tf$reduce_mean(tf$square(var - mean)))
      })
      tf$summary$scalar(paste0("stddev/", name), stddev)
```

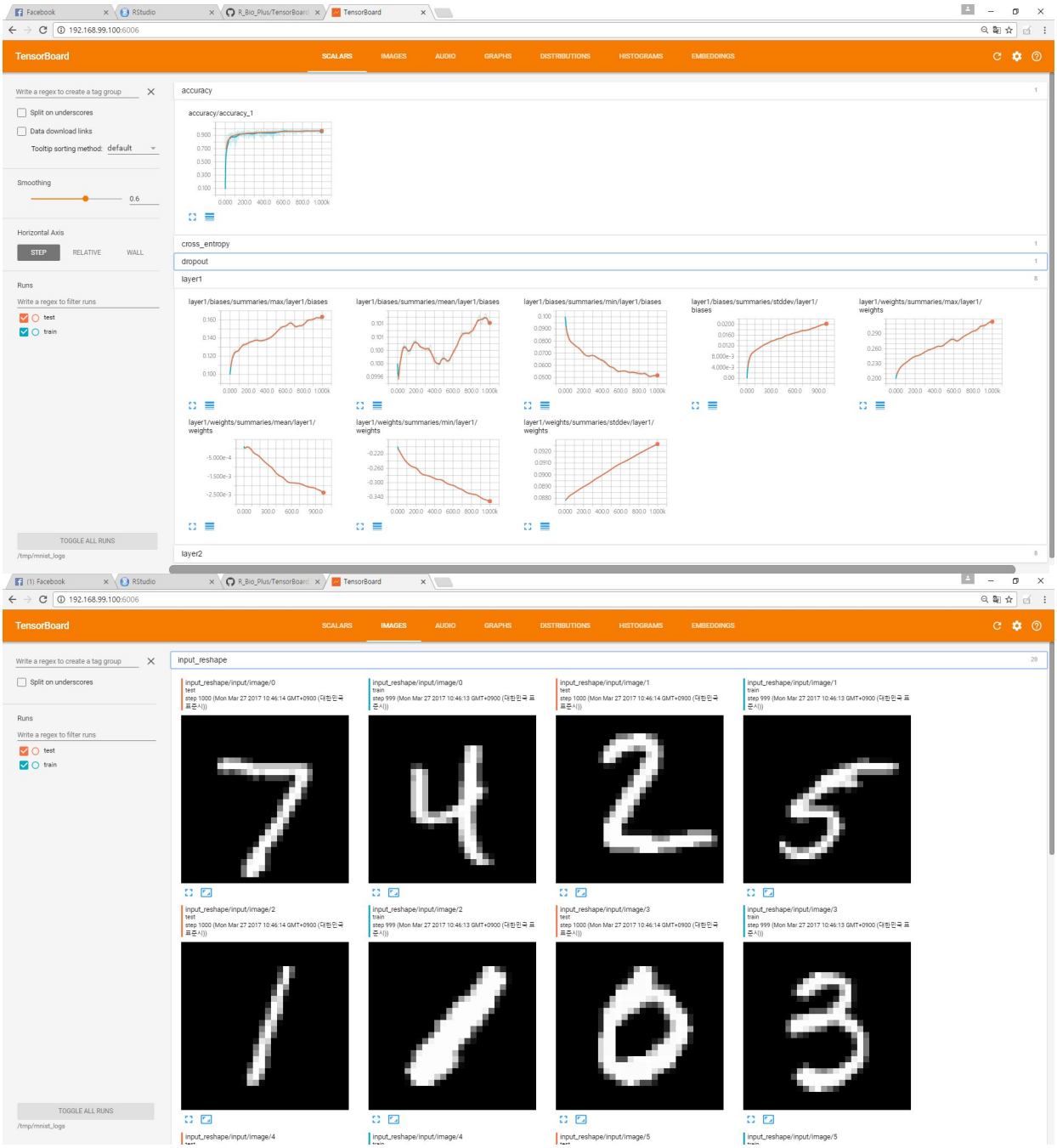
```
      tf$summary$scalar(paste0("max/", name), tf$reduce_max(var))
      tf$summary$scalar(paste0("min/", name), tf$reduce_min(var))
      tf$summary$histogram(name, var)
    })
  }

  # Reusable code for making a simple neural net layer.
  #
  # It does a matrix multiply, bias add, and then uses relu to nonlinearize.
  # It also sets up name scoping so that the resultant graph is easy to rea
d,
  # and adds a number of summary ops.
  #
  nn_layer <- function(input_tensor, input_dim, output_dim,
                       layer_name, act=tf$nn$relu) {
    with(tf$name_scope(layer_name), {
      # This Variable will hold the state of the weights for the layer
      with(tf$name_scope("weights"), {
        weights <- weight_variable(shape(input_dim, output_dim))
        variable_summaries(weights, paste0(layer_name, "/weights"))
      })
      with(tf$name_scope("biases"), {
        biases <- bias_variable(shape(output_dim))
        variable_summaries(biases, paste0(layer_name, "/biases"))
      })
      with (tf$name_scope("Wx_plus_b"), {
        preactivate <- tf$matmul(input_tensor, weights) + biases
        tf$summary$histogram(paste0(layer_name, "/pre_activations"), preacti
vate)
      })
      activations <- act(preactivate, name = "activation")
      tf$summary$histogram(paste0(layer_name, "/activations"), activations)
    })
    activations
  }

  hidden1 <- nn_layer(x, 784L, 500L, "layer1")

  with(tf$name_scope("dropout"), {
    keep_prob <- tf$placeholder(tf$float32)
    tf$summary$scalar("dropout_keep_probability", keep_prob)
    dropped <- tf$nn$dropout(hidden1, keep_prob)
  })

  y <- nn_layer(dropped, 500L, 10L, "layer2", act = tf$nn$softmax)

  with(tf$name_scope("cross_entropy"), {
    diff <- y_ * tf$log(y)
    with(tf$name_scope("total"), {
      cross_entropy <- -tf$reduce_mean(diff)
    })
    tf$summary$scalar("cross entropy", cross_entropy)
```

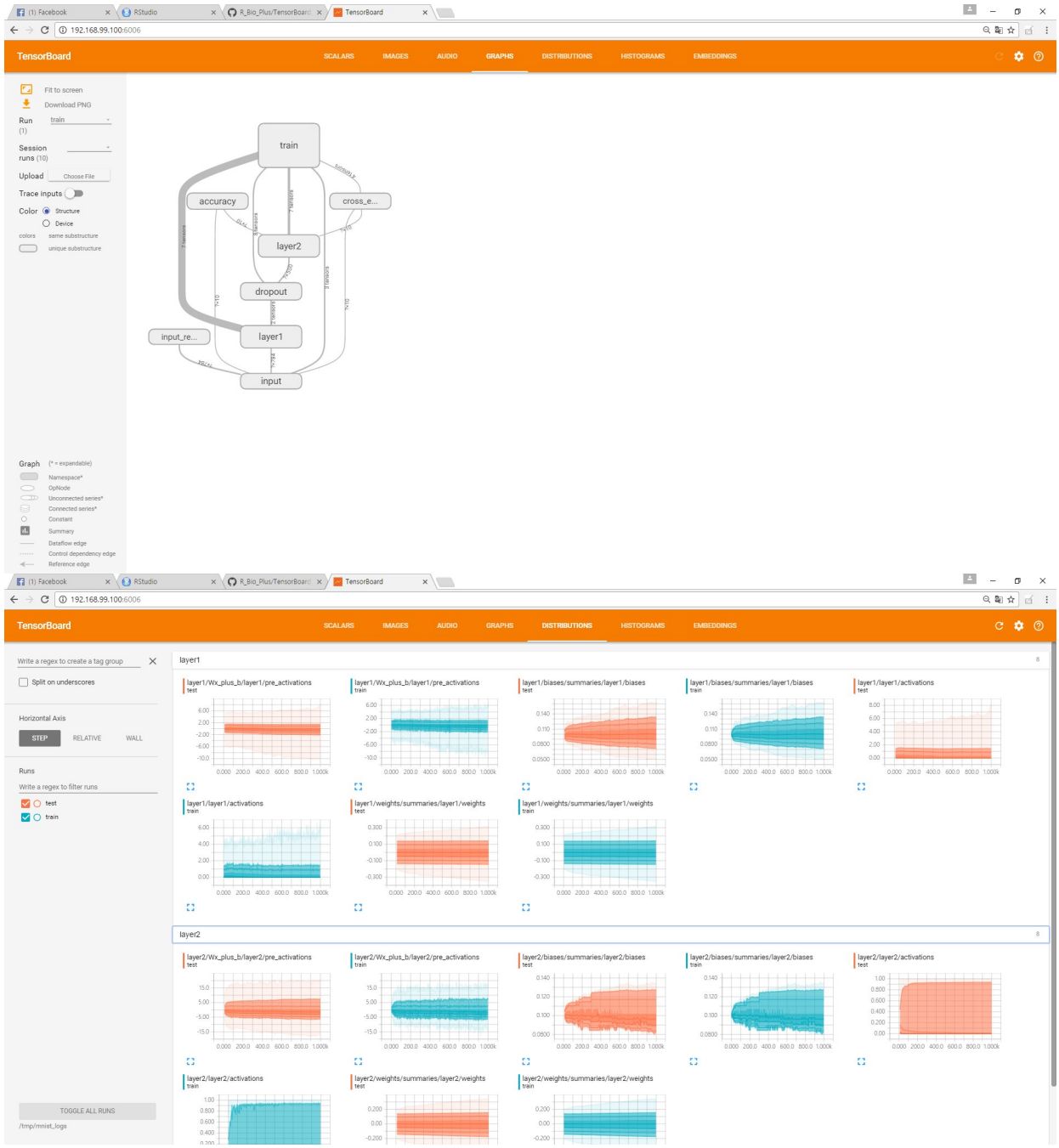
```
  })

  with(tf$name_scope("train"), {
    optimizer <- tf$train$AdamOptimizer(FLAGS$learning_rate)
    train_step <- optimizer$minimize(cross_entropy)
  })

  with(tf$name_scope("accuracy"), {
    with(tf$name_scope("correct_prediction"), {
      correct_prediction <- tf$equal(tf$arg_max(y, 1L), tf$arg_max(y_, 1L))
    })
    with(tf$name_scope("accuracy"), {
      accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))
    })
    tf$summary$scalar("accuracy", accuracy)
  })

  # Merge all the summaries and write them out to /tmp/mnist_logs (by defaul
t)
  merged <- tf$summary$merge_all()
  train_writer <- tf$summary$FileWriter(file.path(FLAGS$summaries_dir, "trai
n"),
                                        sess$graph)
  test_writer <- tf$summary$FileWriter(file.path(FLAGS$summaries_dir, "tes
t"))
  sess$run(tf$global_variables_initializer())

  # Train the model, and also write summaries.
  # Every 10th step, measure test-set accuracy, and write test summaries
  # All other steps, run train_step on training data, & add training summari
es

  # Make a TensorFlow feed_dict: maps data onto Tensor placeholders.
  feed_dict <- function(train) {
    if (train || FLAGS$fake_data) {
      batch <- mnist$train$next_batch(100L, fake_data = FLAGS$fake_data)
      xs <- batch[[1]]
      ys <- batch[[2]]
      k <- FLAGS$dropout
    } else {
      xs <- mnist$test$images
      ys <- mnist$test$labels
      k <- 1.0
    }
    dict(x = xs,
         y_ = ys,
         keep_prob = k)
  }

  for (i in 1:FLAGS$max_steps) {
    if (i %% 10 == 0) { # Record summaries and test-set accuracy
      result <- sess$run(list(merged, accuracy), feed_dict = feed_dict(FALS
```

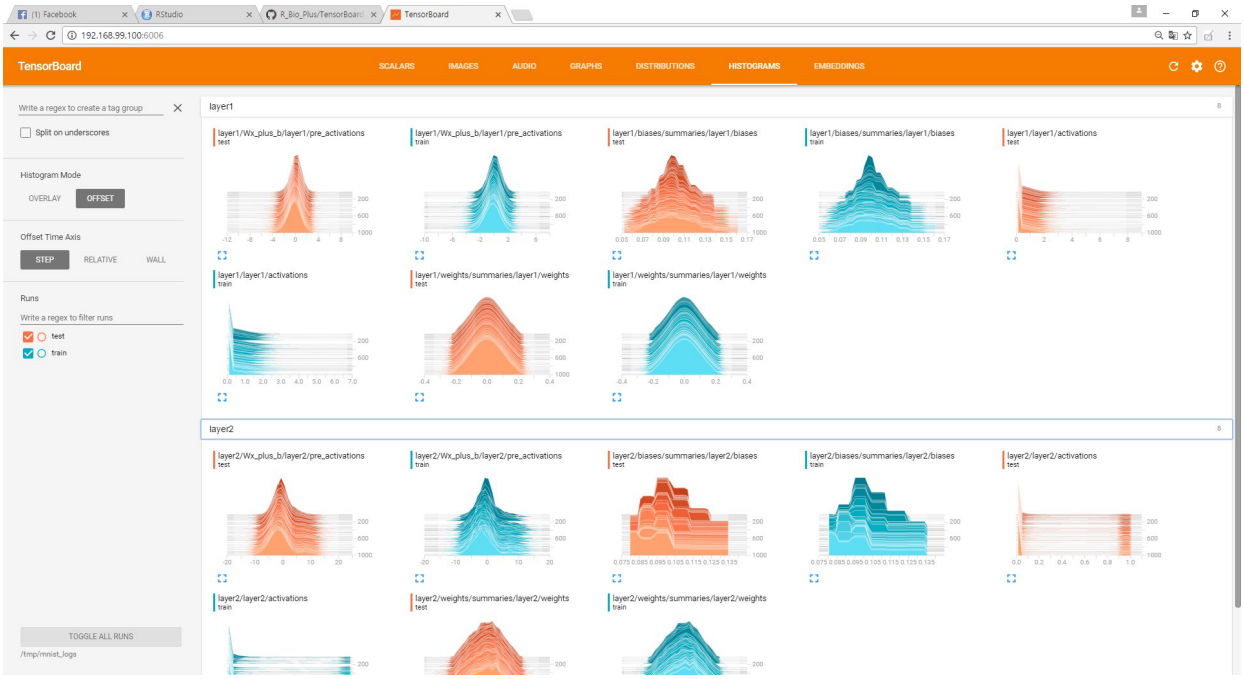
```
E))
      summary <- result[[1]]
      acc <- result[[2]]
      test_writer$add_summary(summary, i)
    } else {  # Record train set summaries, and train
      if (i %% 100 == 99) { # Record execution stats
        run_options <- tf$RunOptions(trace_level = tf$RunOptions()$FULL_TRAC
E)
        run_metadata <- tf$RunMetadata()
        result <- sess$run(list(merged, train_step),
                           feed_dict = feed_dict(TRUE),
                           options = run_options,
                           run_metadata = run_metadata)
        summary <- result[[1]]
        train_writer$add_run_metadata(run_metadata, sprintf("step%03d", i))
        train_writer$add_summary(summary, i)
        cat("Adding run metadata for ", i, "\n")
      } else {  # Record a summary
        result <- sess$run(list(merged, train_step), feed_dict = feed_dict(T
RUE))
        summary <- result[[1]]
        train_writer$add_summary(summary, i)
      }
    }
  }

  train_writer$close()
  test_writer$close()
}

# initialize summaries_dir (remove existing if necessary)
if (tf$gfile$Exists(FLAGS$summaries_dir))
  tf$gfile$DeleteRecursively(FLAGS$summaries_dir)
tf$gfile$MakeDirs(FLAGS$summaries_dir)

# train
# train()
```

# Launching TensorBoard

- tensorboard –logdir=/tmp/mnist_logs
- http://192.168.99.100:6006/ (http://192.168.99.100:6006/) 로 접속


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
accuracy
cross_entropy
dropout
layer1
layer2
input_reshape
Runs
test
train
TOGGLE ALL RUNS


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
train
accuracy
cross_e...
layer2
dropout
layer1
input_re...
input
TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
layer1
layer2
STEP
RELATIVE
WALL
TOGGLE ALL RUNS


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
Write a regex to create a tag group
Split on underscores
Histogram Mode
OVERLAY
OFFSET
Offset Time Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
test
train
TOGGLE ALL RUNS
/tmp/mnist_logs
layer1
layer1/Wx_plus_b/layer1/pre_activations
layer1/biases/summaries/layer1/biases
layer1/layer1/activations
layer1/weights/summaries/layer1/weights
layer2
layer2/Wx_plus_b/layer2/pre_activations
layer2/biases/summaries/layer2/biases
layer2/layer2/activations
layer2/weights/summaries/layer2/weights