6 - Tensorflow and Tensorboard

Ludwig Krippahl

Tensorboard

Summary

- Graph mode and Tensorboard
- Numerical stability
- Tutorial exercises:

- Regression (Auto MPG)
- Multiclass classification (MNIST)

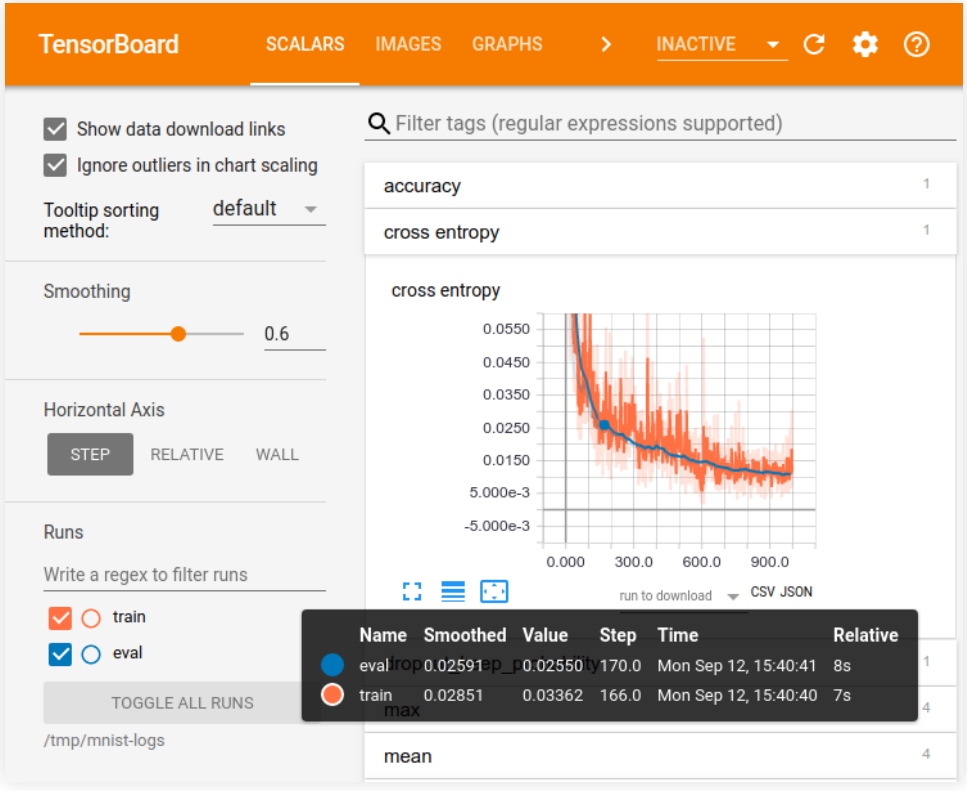
Tensorboard

Tensorboard

Tensorboard is a suite of visualization tools

- Useful for monitoring the learning process

- (More info in Tensorflow programmers' guide)



Tensorboard

How to use Tensorboard:

- In your code, create a logging directory for each run of each model:

```
import tensorflow as tf
import numpy as np
from datetime import datetime
now = datetime.utcnow().strftime("%Y%m%d%H%M%S")
root_logdir = "logs"
log_dir = "{}/model-{}/".format(root_logdir, now)
```

- Now we need to create the computation graph.

How to use Tensorboard:

- Let's use the logistic regression model

- Note the reshape, so output is a 1D vector

```
weights = tf.Variable(tf.random.normal((2,1)))
bias = tf.Variable(0.0)

def prediction(X):
    net = tf.add(tf.matmul(X, weights), bias, name="net")
    return tf.reshape(tf.nn.sigmoid(net, name="output"),[-1])

def logistic_loss(predicted,Y):
    cost = -tf.reduce_mean(Y * tf.math.log(predicted) + (1-Y) *\
                           (tf.math.log(1-predicted)))
    return cost

def grad(X, y):
    with tf.GradientTape() as tape:
        predicted = prediction(X)
        loss_val = logistic_loss(predicted,y)
    return tape.gradient(loss_val, [weights, bias]),[weights,bias]
```

Tensorboard

- Now we create the graph, using AutoGraph

- AutoGraph is implemented in the `tf.function` wrapper

```
@tf.function
def create_graph(X,Y):
    prediction(X)

def write_graph(X,Y,writer):
    tf.summary.trace_on(graph=True)
    create_graph(tf.constant(X.astype(np.float32)),
                 tf.constant(Y.astype(np.float32)))
    with writer.as_default():
        tf.summary.trace_export(name="trace",step=0)

mat = np.loadtxt('gene_data.txt',delimiter='\t')
Ys = mat[:,-1]
Xs = mat[:,:-1]
means = np.mean(Xs,0)
stdevs = np.std(Xs,0)
Xs = (Xs-means)/stdevs
writer = tf.summary.create_file_writer(log_dir)
write_graph(Xs,Ys,writer)
```

Tensorboard

How to use Tensorboard:

- Now we need to create the graph, using AutoGraph
  - AutoGraph is implemented in the `tf.function` wrapper
- We will not be looking at AutoGraph in detail
  - It is useful for performance because Tensorflow can optimize the code
  - But requires more detailed understanding of Tensorflow
- You can read more on the documentation:
  - https://www.tensorflow.org/guide/function
- Here we will use AutoGraph only for representing the graph on Tensorboard

Tensorboard

Running tensorboard:

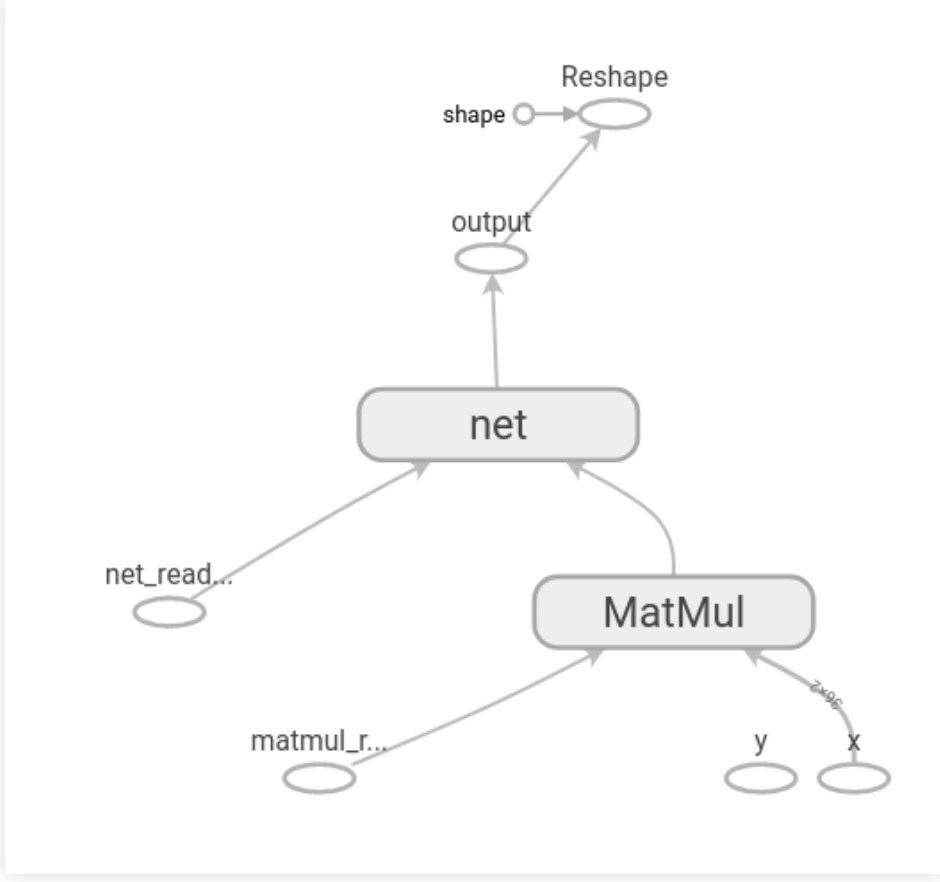
- In the shell prompt, type

```
tensorboard --logdir [your log dir]
```

- Open the browser on URL `http://127.0.0.1:6006`

Tensorboard

- The activation computation graph:



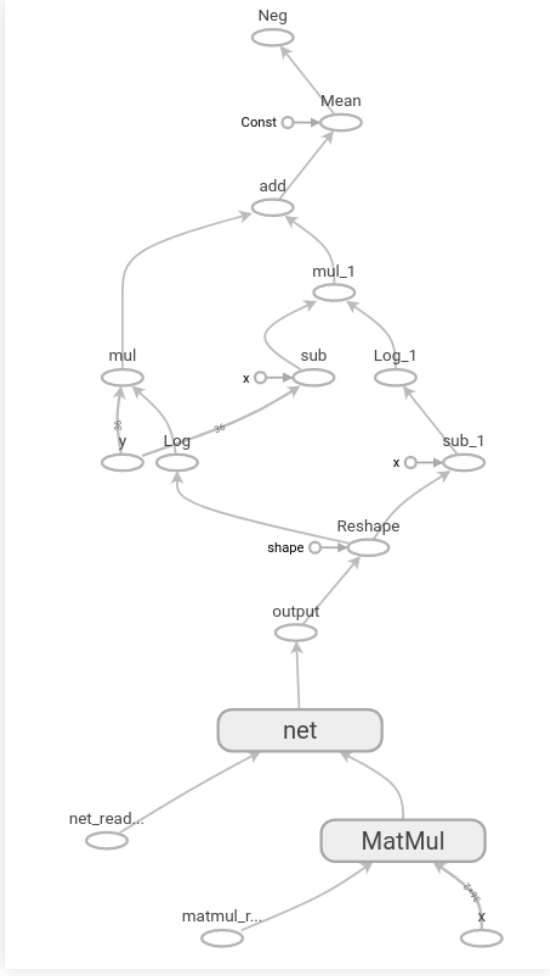
Tensorboard

- We can include the loss function too:

```
@tf.function
def create_graph(X,Y):
    predicted = prediction(X)
    logistic_loss(predicted,Y)

def write_graph(X,Y,writer):
    tf.summary.trace_on(graph=True)
    create_graph(tf.constant(X.astype(np.float32)),
                 tf.constant(Y.astype(np.float32)))
    with writer.as_default():
        tf.summary.trace_export(name="trace",step=0)

mat = np.loadtxt('gene_data.txt',delimiter='\t')
Ys = mat[:,-1]
Xs = mat[:,:-1]
means = np.mean(Xs,0)
stdevs = np.std(Xs,0)
Xs = (Xs-means)/stdevs
writer = tf.summary.create_file_writer(log_dir)
write_graph(Xs,Ys,writer)
```

Tensorboard

- Activation and loss function:



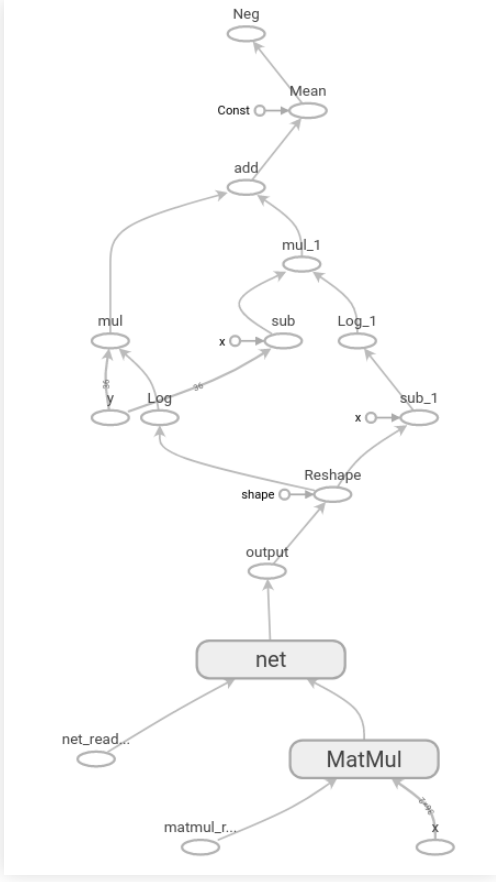
Tensorboard

- Or the whole graph, including the gradients for backpropagation

```
def grad(X, y):
    with tf.GradientTape() as tape:
        predicted = prediction(X)
        loss_val = logistic_loss(predicted,y)
    return tape.gradient(loss_val, [weights, bias]),[weights,bias]

@tf.function
def create_graph(X,Y):
    grad(X,Y)

def write_graph(X,Y,writer):
    tf.summary.trace_on(graph=True)
    create_graph(tf.constant(X.astype(np.float32)),
                 tf.constant(Y.astype(np.float32)))
    with writer.as_default():
        tf.summary.trace_export(name="trace",step=0)

...
writer = tf.summary.create_file_writer(log_dir)
write_graph(Xs,Ys,writer)
```

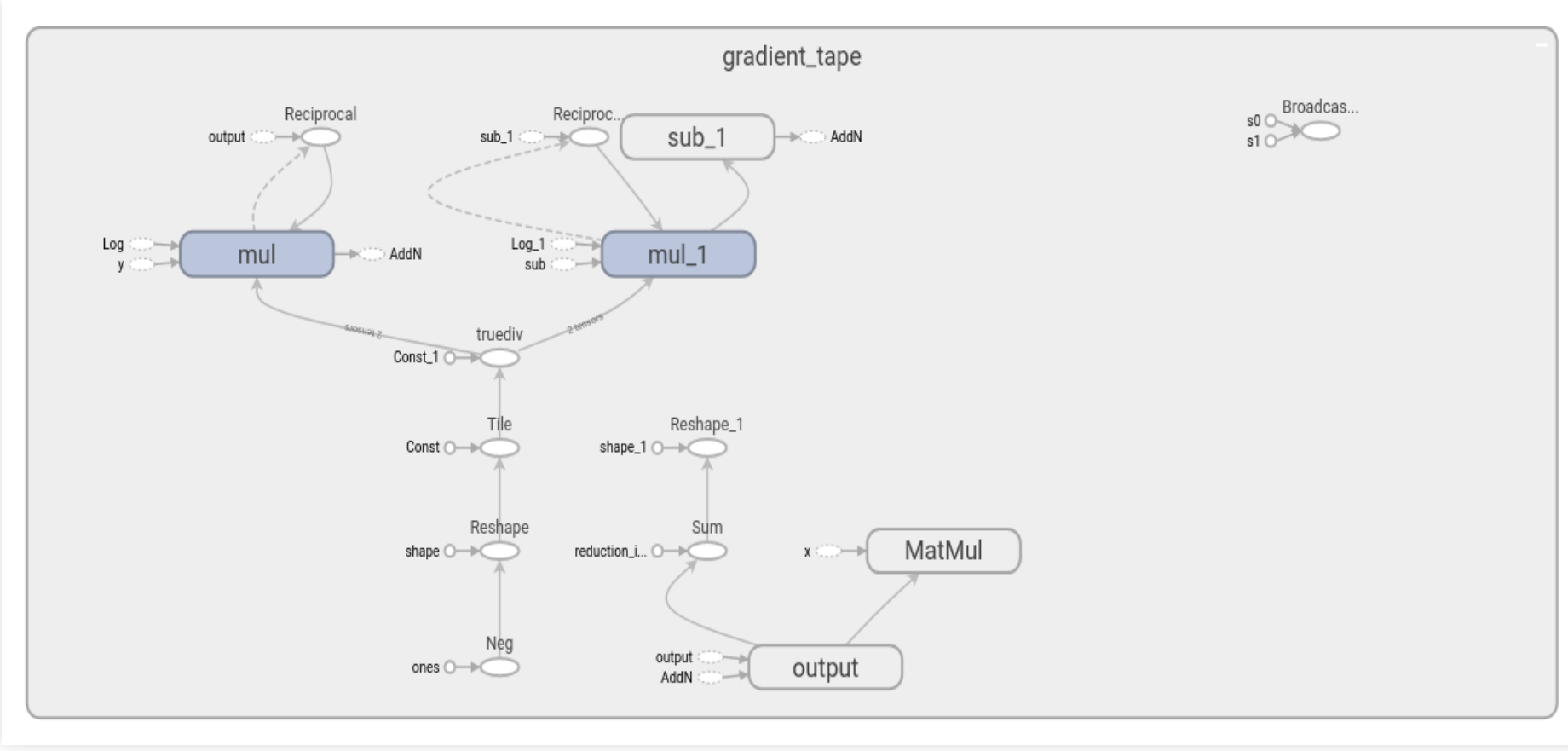
Tensorboard

- Same as previous



Tensorboard

- Same as previous, but also with gradients:



Tensorboard

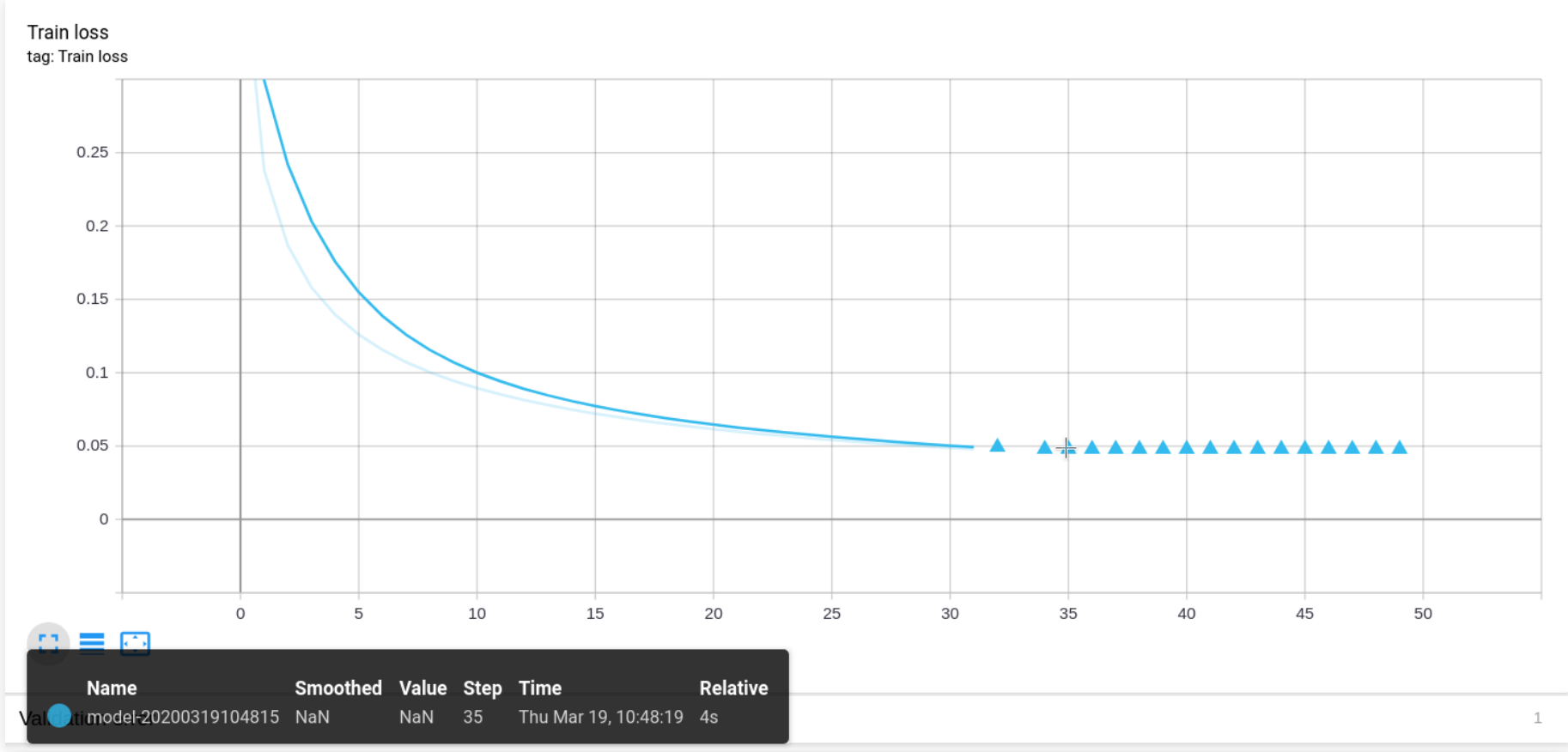
How to use Tensorboard:

- We can add values to Tensorboard

```
learning_rate = 0.01
batch_size = 1
batches_per_epoch = Xs.shape[0]//batch_size
epochs = 50

for epoch in range(epochs):
    shuffled = np.arange(len(Ys))
    np.random.shuffle(shuffled)
    for batch_num in range(batches_per_epoch):
        start = batch_num*batch_size
        batch_xs = tf.constant(Xs[shuffled[start:start+batch_size],:].astype(np
        batch_ys = tf.constant(Ys[shuffled[start:start+batch_size]].astype(np.f
        gradients,variables = grad(batch_xs, batch_ys)
        optimizer.apply_gradients(zip(gradients, variables))
    y_pred =prediction(tf.constant(Xs.astype(np.float32)))
    loss = logistic_loss(y_pred,Ys)
    print(f"Epoch {epoch}, loss {loss}")
    with writer.as_default():
        tf.summary.scalar('Train loss', loss, step=epoch)
```

Tensorboard

- Now we can see the results (oops)



Tensorboard

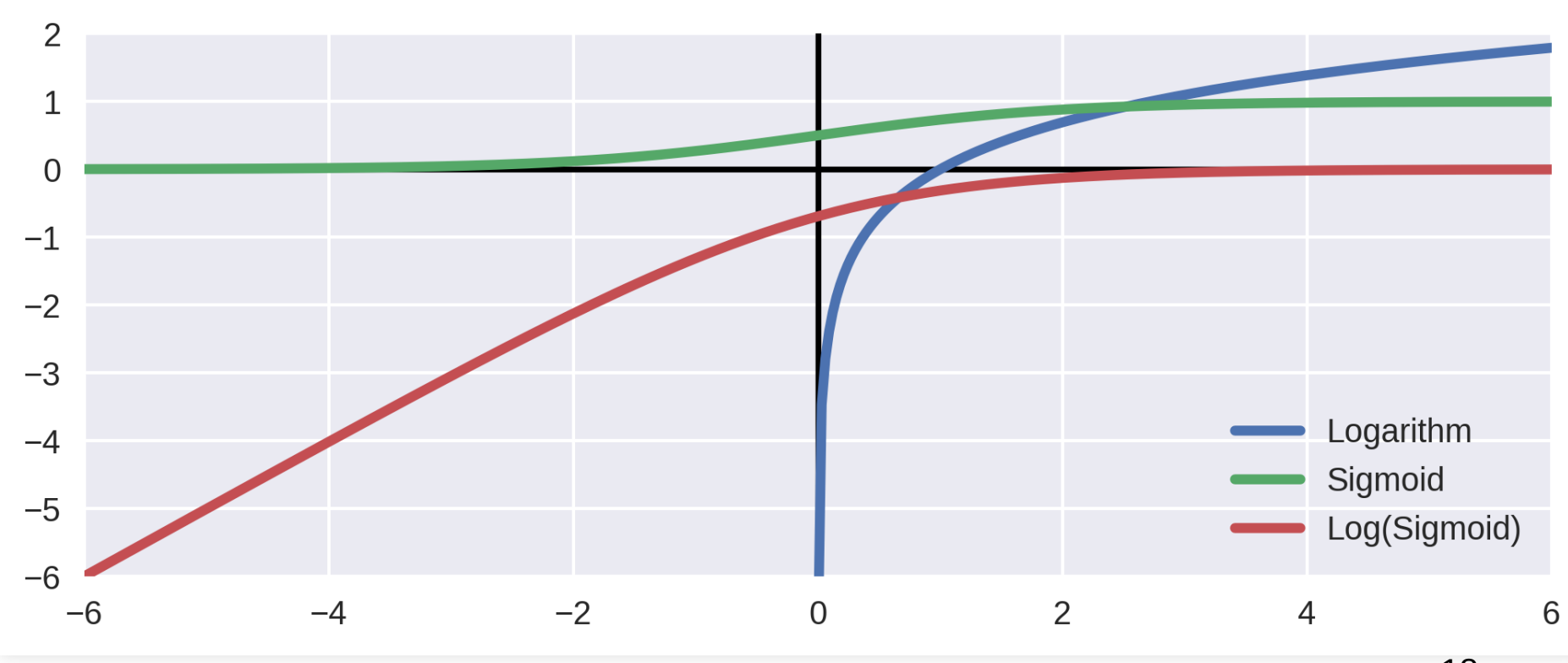
- Something is wrong

```
Epoch 23, loss 0.0518291071057319 64
Epoch 24, loss 0.0506272092461586
Epoch 25, loss 0.0495013855397701 26
Epoch 26, loss nan
Epoch 27, loss nan
Epoch 28, loss 0.0464570112526416 8
Epoch 29, loss 0.0455445311963558 2
Epoch 30, loss nan
Epoch 31, loss nan
Epoch 32, loss nan
Epoch 33, loss nan
Epoch 34, loss nan
Epoch 35, loss nan
```

Tensorboard

- The problem is in computing the logistic loss

```
def logistic_loss(predicted,Y):
    cost = -tf.reduce_mean(Y * tf.math.log(predicted) + (1-Y) *\
                           (tf.math.log(1-predicted)))
    return cost
```



Tensorboard

- Some solutions:

- (Machler 2012)

$$\log(s(x)) = \begin{cases} x & x < -33.3 \\ x - \exp(x) & -33.3 \leq x \leq -18 \\ -\text{log1p}(\exp(-x)) & -18 \leq x \leq 37 \\ -\exp(-x) & 37 \leq x \end{cases}$$

- Tensorflow:

```
loss = max(x, 0) - x * y + log(1 + exp(-abs(x)))
```

- Where x is the input to the sigmoid function and y the target label.

Tensorboard

- We will use the Tensorflow implementation of sigmoid cross entropy

- This function receives as argument the linear output of the neuron (the *logits*)
- Note: `grad` now just calls `logistic_loss`

```
def prediction(X):
    net = tf.add(tf.matmul(X, weights), bias, name="net")
    return tf.reshape(tf.nn.sigmoid(net, name="output"),[-1])

def logits(X):
    net = tf.add(tf.matmul(X, weights), bias, name="net")
    return tf.reshape(net,[-1])

def logistic_loss(X,Y):
    net = logits(X)
    cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(Y,net))
    return cost

def grad(X, Y):
    with tf.GradientTape() as tape:
        loss_val = logistic_loss(X,Y)
    return tape.gradient(loss_val, [weights, bias]),[weights,bias]
```

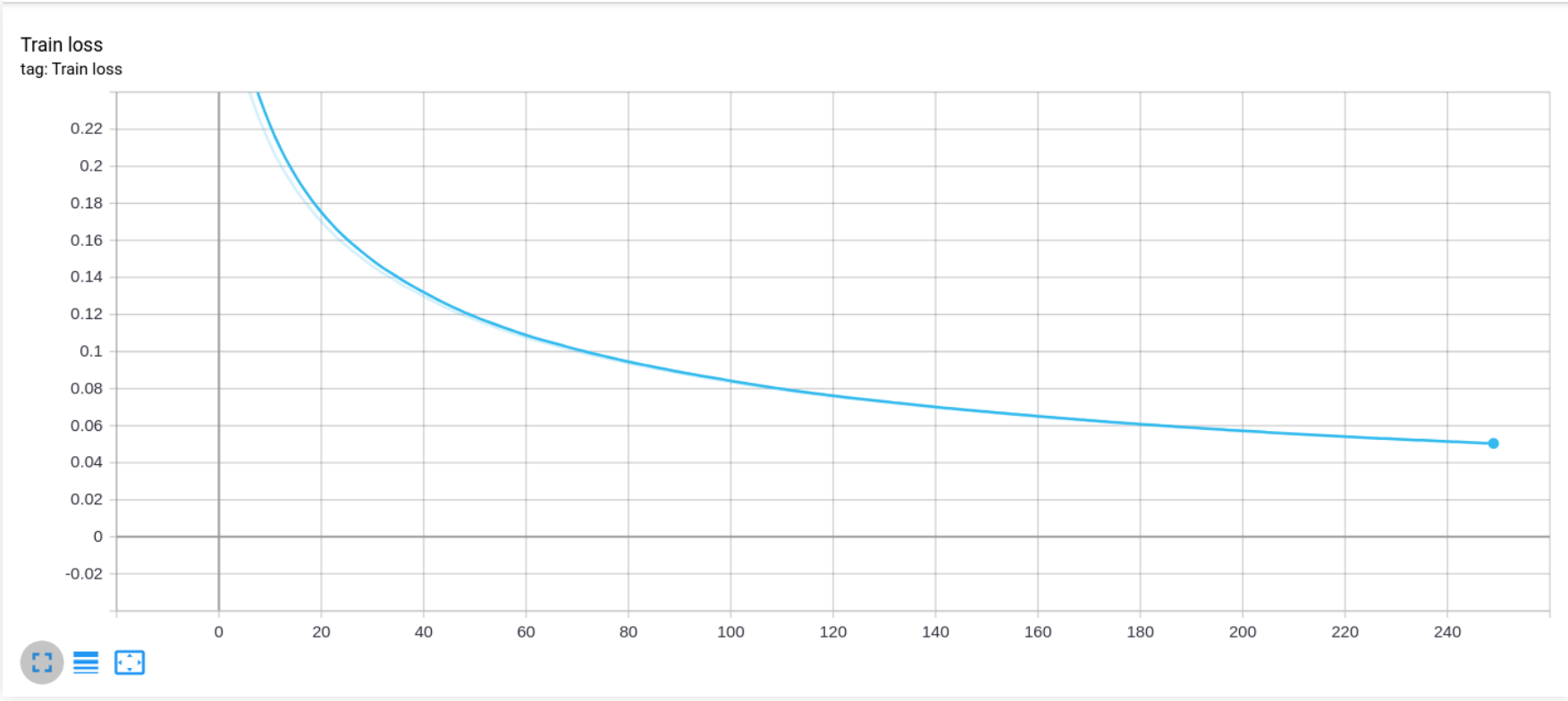
Tensorboard

- We need to change the training loop too

```
for epoch in range(epochs):
    shuffled = np.arange(len(Ys))
    np.random.shuffle(shuffled)
    for batch_num in range(batches_per_epoch):
        start = batch_num*batch_size
        batch_xs = tf.constant(Xs[shuffled[start:start+batch_size],:].astype(np
        batch_ys = tf.constant(Ys[shuffled[start:start+batch_size]].astype(np.f
        gradients,variables = grad(batch_xs, batch_ys)
        optimizer.apply_gradients(zip(gradients, variables))
    loss = logistic_loss(tf.constant(Xs.astype(np.float32)),
                         tf.constant(Ys.astype(np.float32)))
    print(f"Epoch {epoch}, loss {loss}")
    with writer.as_default():
        tf.summary.scalar('Train loss', loss, step=epoch)
```

Tensorboard

- And now it works



Loss functions in Tensorflow

- `tensorflow.losses.mean_squared_error`

- The mean squared error loss function. This one is not so important (no problems with squared errod)

- `tensorflow.losses.sigmoid_cross_entropy`

- The cross entropy function for a sigmoid output. Argument is logits

- `tensorflow.losses.softmax_cross_entropy`

- The cross entropy function for a softmax output. Also logits

Example: Regression with MLP

Regression with the Auto MPG Data Set

- Predict the fuel consumption of cars

- Auto MPG Data Set, available at the UCI repository

- Some pointers:

- Shuffle the data before splitting into training and validation sets
- Target: first column, MPG (miles per gallon)
- Standardize features and target value
- Standardizing target centers the output and minimizes numerical problems
- Only 392 examples. Use 300 for training and 92 for validation.
- Linear output for he last neuron (only one neuron)
- Mean squared error loss function for training
- Try different activations and learning parameters (rate, momentum)

Regression with the Auto MPG Data Set

- Load, standardize, use 300 for train and rest for validation:

```
data = np.loadtxt('AutoMPG.tsv',skiprows=1)
np.random.shuffle(data)
means = np.mean(data,axis=0)
stds = np.std(data,axis=0)
data = (data-means)/stds

valid_Y = data[300:,0]
valid_X = data[300:,1:]

Y = data[:300,0]
X = data[:300,1:]
```

Regression

- Create the model

```
def layer(inputs,neurons):
    weights = tf.Variable(tf.random.normal((inputs.shape[1],neurons)) )
    bias = tf.Variable(tf.zeros([neurons]))
    return weights,bias

def create_network(X,layers):
    network = []
    variables = []
    previous = X
    for ix, neurons in enumerate(layers):
        weights,bias = layer(previous,neurons)
        network.append( (weights,bias) )
        variables.extend( (weights,bias) )
        previous = weights
    return network, variables
```

- We can use these functions in this way:

```
layers = [25,15,1]
network,variables = create_network(X,layers)
```

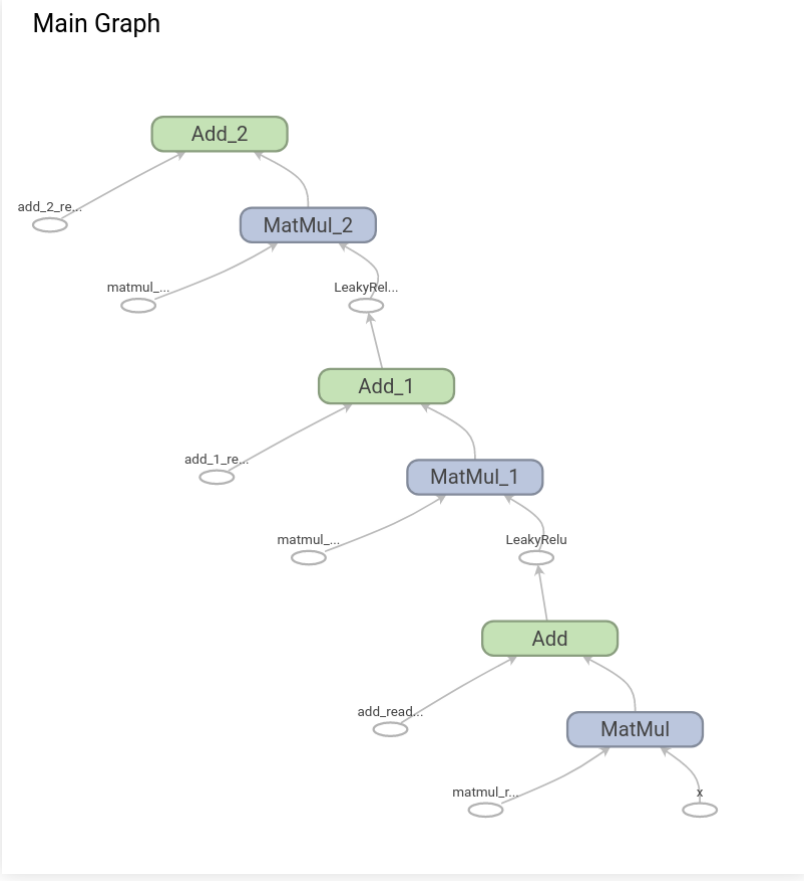
Regression

- Create the model, and save graph

```
def predict(X):
    net = X
    for weights,bias  in network[:-1]:
        net = tf.add(tf.matmul(net, weights), bias)
        net = tf.nn.leaky_relu(net)
    weights,bias = network[-1]
    net = tf.add(tf.matmul(net, weights), bias)
    return tf.reshape(net,[-1])

@tf.function
def create_graph(X):
    predict(X)

def write_graph(X):
    tf.summary.trace_on(graph=True)
    create_graph(tf.constant(X.astype(np.float32)))
    with writer.as_default():
        tf.summary.trace_export(name="trace",step=0)
```

Regression

- Create the model, and save graph



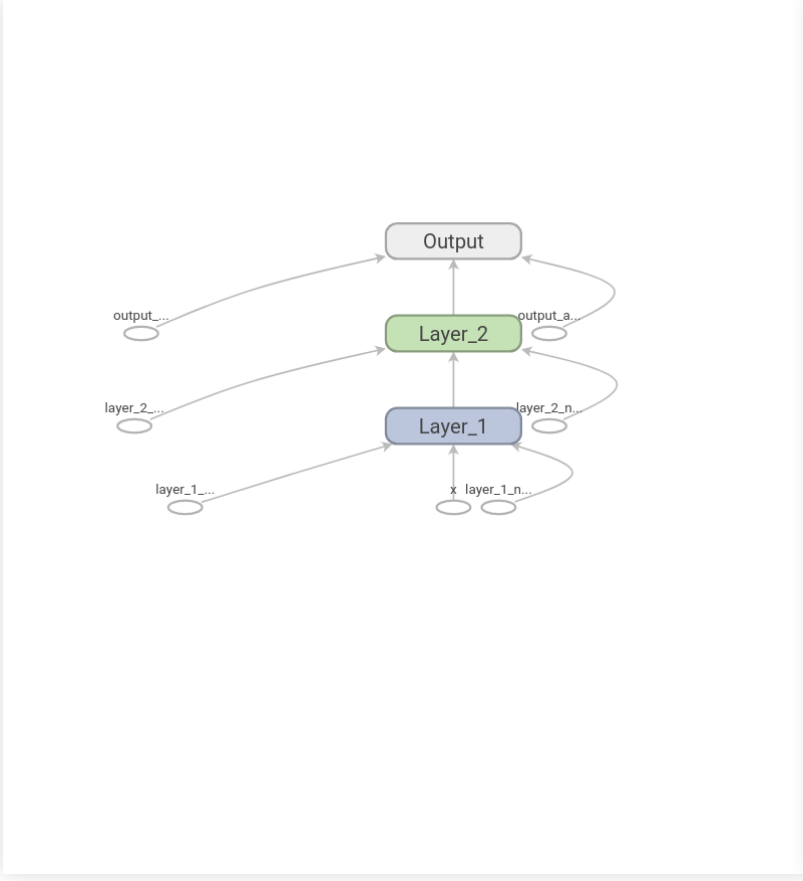
Regression

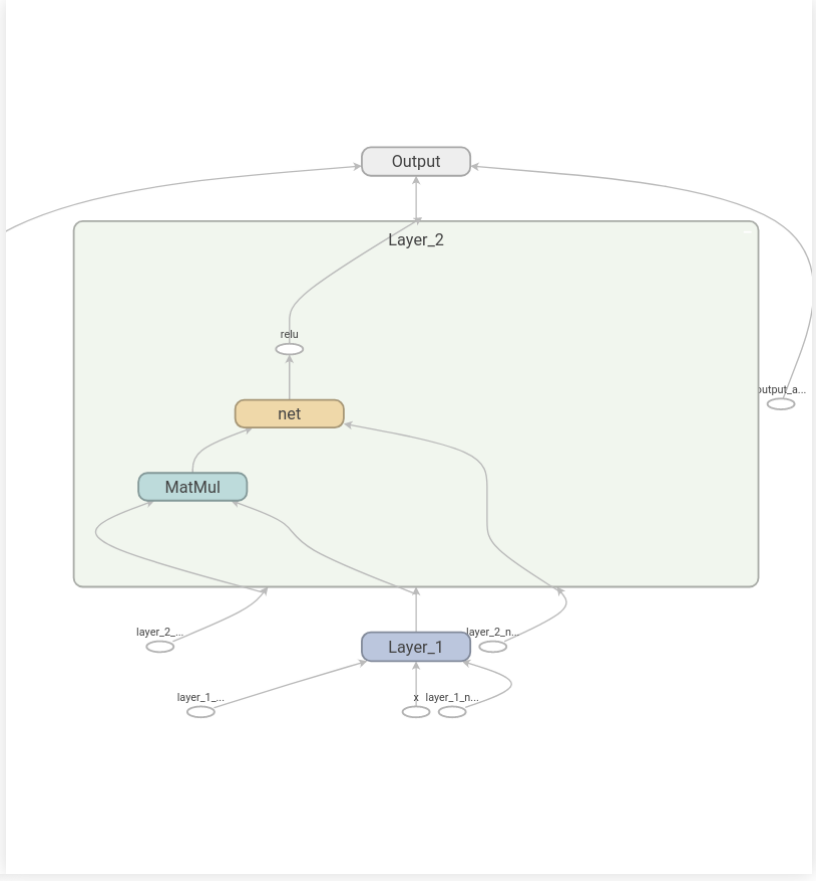
- Better still, use names and name scopes for the operations:

```
def predict(X):
    net = X
    layer = 1
    for weights,bias  in network[:-1]:
        with tf.name_scope(f'Layer_{layer}'):
            net = tf.add(tf.matmul(net, weights), bias,name='net')
            net = tf.nn.leaky_relu(net, name="relu")
        layer += 1
    weights,bias = network[-1]
    with tf.name_scope('Output'):
        net = tf.add(tf.matmul(net, weights), bias)
    return tf.reshape(net,[-1])
```

Regression

- Better still, use names and name scopes for the operations:





Numerical problems, again

- Suggested parameters for training

- SGD optimizer with a learning rate of 0.005 and momentum of 0.9
- Batch size of 32.
- Remember the squared error loss function and no activation on the output

- If many neurons, this will not work (NaN again)
- This time the problem is with weight initialization (but try anyway)

```
def layer(inputs,neurons,layer_name):
   weights = tf.Variable(tf.random.normal((inputs.shape[1],neurons),
                         stddev = 1/neurons ))
   bias = tf.Variable(tf.zeros([neurons]))
   return weights,bias
```

- (More on this next week)

Now it should work

- Convert squared error into MPG:

```
...
train_error = loss(tf.constant(X.astype(np.float32)),
                   tf.constant(Y.astype(np.float32)))**0.5*stds[0]
valid_error = loss(tf.constant(valid_X.astype(np.float32)),
                   tf.constant(valid_Y.astype(np.float32)))**0.5*stds[0]
with writer.as_default():
    tf.summary.scalar('Train error', train_error, step=epoch)
    tf.summary.scalar('Validation error', valid_error, step=epoch)
...
```

- Try different architectures
- Monitor train and validation to detect overfitting
- Fine tune learning parameters

Exercice: Softmax for multi-class

Classify MNIST with Softmax

- Modified NIST (MNIST) dataset, handwritten digits

Classify MNIST with Softmax

- Download data (once) and read it

```
import tensorflow as tf
import numpy as np
from tensorflow import keras

mnist  = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

- Reshape from 28x28 to 784 features for each example
- Split train set 50000 for training, 10000 for validation
- Use `layer` function if you want
- Use `tf.nn.softmax_cross_entropy_with_logits` for loss

- Remember: it uses logits, and not the softmax activation

Softmax

- One-hot encoding for encoding multi-class.

```
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
...
```

- Softmax will output vectors with sum of 1 and with probabilities of belonging to each class

- Use `tf.keras.utils.to_categorical` to convert labels.

```
valid_images = train_images[50000:].reshape(-1,28*28)
valid_labels = tf.keras.utils.to_categorical(train_labels[50000:])
train_images = train_images[:50000].reshape(-1,28*28)
train_labels = tf.keras.utils.to_categorical(train_labels[:50000])
```

- Use `np.argmax` if you want to measure accuracy, to find the chosen class

Classify MNIST with Softmax, objective

- Compare different activation functions in the hidden layers
  - Sigmoid, ReLU, Leaky ReLU, ...
- Try different network dimensions and different optimizer parameters
- Remember to check overfitting wih validation data
- In the end, train the final network using the complete training set and evaluate its performance on the test set.
- Note: This time the prediction output must be a matrix
  - Predictions and labels are one-hot encoded
- Check yann.lecun.com/exdb/mnist

Tensorboard

Summary

- Graph mode and Tensorboard
- Numerical stability
- Tutorial exercises:

- Binary classification: gene data with one sigmoid neuron
- Regression: Miles per gallon, linear output
- Multiclass classification: MNIST, 10 classes, softmax output