Predicting the Survival on the Titanic

吳易倫

Outline

- Problem
- Dataset
- Package Introduction
- Tasks
  - Data Preprocessing
  - Build Deep Neural Network
  - Run the Operation
  - Visualization

Problem

Survive or die?

Dataset

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, M...	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, ...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen...	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle,...	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr...	male	35.0	0	0	373450	8.0500	NaN	S
...												

- 892 people
- 12 attributes

survived - Survival (0 = No; 1 = Yes)
class - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
name - Name
sex - Sex
age - Age
sibsp - Number of Siblings/Spouses Aboard
parch - Number of Parents/Children Aboard
ticket - Ticket Number
fare - Passenger Fare
cabin - Cabin
embarked - Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Package Introduction

- TensorFlow 1.13
  - An end-to-end open source platform for machine learning
- tqdm
  - A fast, extensible **progress bar** for Python and CLI
- scikit-learn
  - Evaluation tools
- numpy
  - A powerful N-dimensional array object
  - Sophisticated functions
- matplotlib
  - A Python 2D plotting library

Tasks

Get source code

- SSH into your server and find any good place

```
$ ssh nctuece@[your server ip]
```

- Clone the source code from GitHub

```
$ git clone https://github.com/w86763777/HCC-ML-LAB
$ cd HCC-ML-LAB/LAB1
```

Setup Environment

- Create virtual environment
  - Install pip

```
$ sudo apt-get install python3-pip
```

  - Install virtualenv using pip3

```
$ sudo pip3 install virtualenv
```

  - Create a virtual environment

  At the root of your project (i.e. HCC-ML-LAB/LAB2)

```
$ virtualenv -p python3 venv
...
done.
```

    - -p specify python interpreter
    - “venv” is your environment name

Setup Environment

- Activate virtual environment

```
$ source venv/bin/activate
(venv) $
```

- Install packages at a time

```
(venv) $ pip install –r requirements.txt
...
Successfully installed ...
```

- Leave virtual environment

```
(venv) $ deactivate
$
```

Setup Environment

- In requirements.txt

```
flake8==3.7.7
kiwisolver==1.0.1
matplotlib==3.0.3
...
```

- What is requirements.txt ?
  - You don’t have to manually type pip install several times to get all of your packages installed
  - You don’t have to worry about getting the right version installed

Note. How to create requirements.txt ?

```
(venv) $ pip freeze > requirements.txt
```

1-1 Data Preprocessing

- Dataset
  1. https://drive.google.com/open?id=1vkkLO49h6Gk_V4bVAAUJixURWjfxPm4e
  2. Use download script to facilitate the progress. Just run

```
(venv) $ python download.py
```

- You should see

```
(venv) $ ls
download.py  LAB2.py  requirements.txt  titanic.csv  venv
```

1-1 Data Preprocessing

- Read .csv file
  `dataset` is an object containing all the data.

```
import pandas as pd
dataset = pd.read_csv(path)
print(dataset.head(5))
```

pandas.read_csv:
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

1-1 Data Preprocessing

- Replace missing values using the mean along each column.

```
nan_columns = ["Age", "SibSp", "Parch"]
for column in nan_columns:
    imputer = SimpleImputer()
    dataset[column] = imputer.fit_transform(
        dataset[column].values.reshape(-1, 1))
```

`sklearn.impute.`**`SimpleImputer`**(*missing_values=nan*, *strategy='mean'*)

SimpleImputer:
https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

1-1 Data Preprocessing

- One-hot encoding

```
# expand categorical data to one hot format
print(dataset.head(5))
dummy_columns = ["Pclass"]
for column in dummy_columns:
    dataset = pd.concat(
        [dataset, pd.get_dummies(dataset[column], prefix=column)], axis=1)
    dataset = dataset.drop(column, axis=1)
```

```
Survived     Sex   Age  SibSp  Parch. Pclass
       0    male  22.0    1.0    0.0.      3

Survived     Sex   Age  SibSp  Parch  Pclass_1  Pclass_2  Pclass_3
       0    male  22.0    1.0    0.0         0         0         1
```

pandas.get_dummies:
http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

1-1 Data Preprocessing

- Drop not concerned columns

```
# TODO: Checkpoint 1
# 1. Drop not concerned columns
not_concerned_columns = [
    "PassengerId", "Name", "Ticket", "Cabin", "Embarked"]
dataset = dataset.drop(labels=???, axis=???)
```

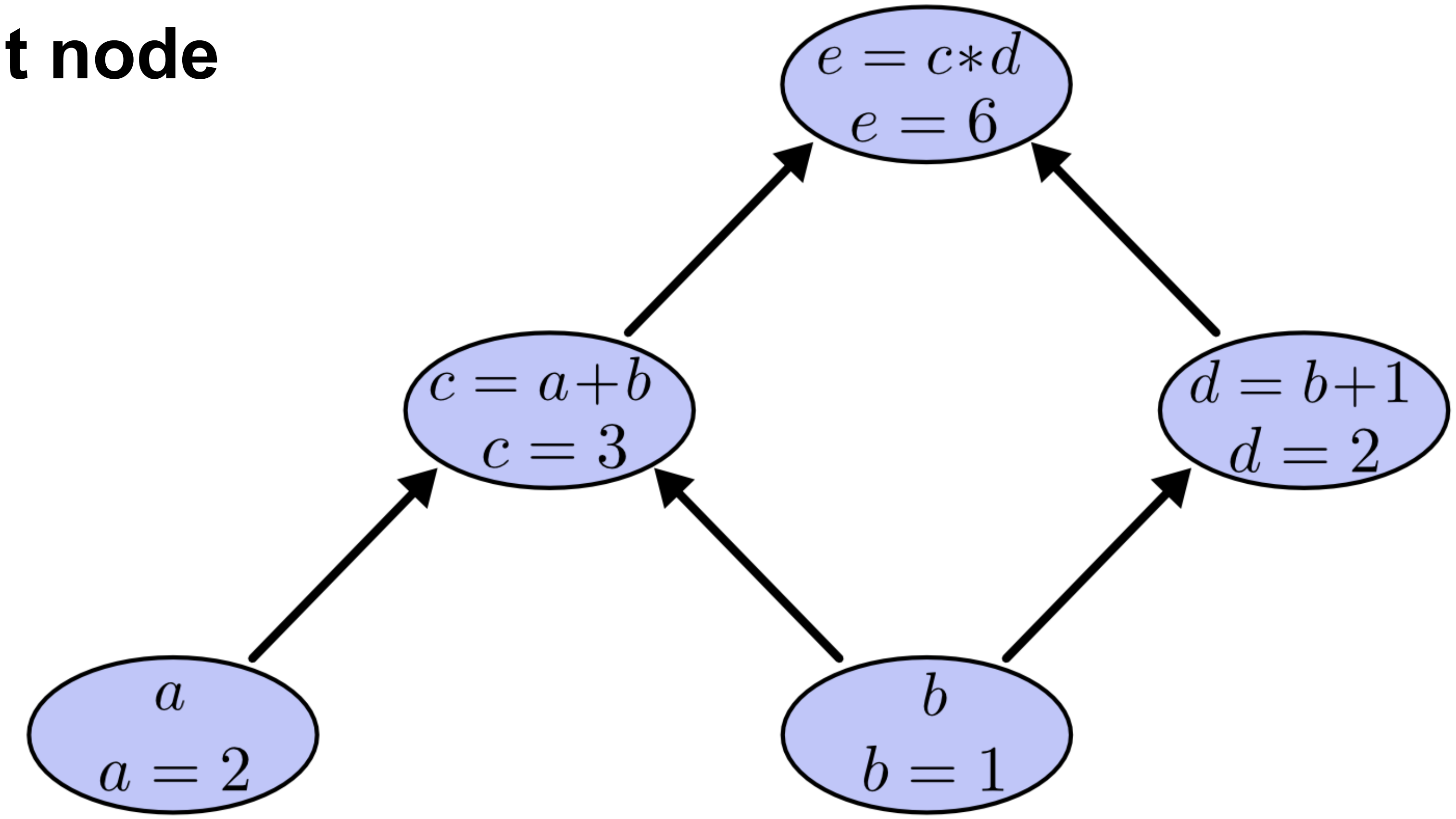
pandas.DataFrame.drop:
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html

Checkpoint 1:
Run LAB2.py and show that

	Survived	Sex	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3
0	0	1	0.271174	1.0	0.0	7.2500	0	0	1
1	1	0	0.472229	1.0	0.0	71.2833	1	0	0
2	1	0	0.321438	0.0	0.0	7.9250	0	0	1
3	1	0	0.434531	1.0	0.0	53.1000	1	0	0
4	0	1	0.434531	0.0	0.0	8.0500	0	0	1

TensorFlow

- Most of Machine Learning computation can be represented by computation graph
- All the trainable variables are attached to “default graph” by default, so reset the default graph at first
- In TensorFlow 1.x, default operation mode is static graph.
- **Create input node -> Create graph (Create output node) -> Create session -> Run specific output node**



TensorFlow

- **Create input node** -> Create graph (Create output node) -> Create session -> Run specific output node

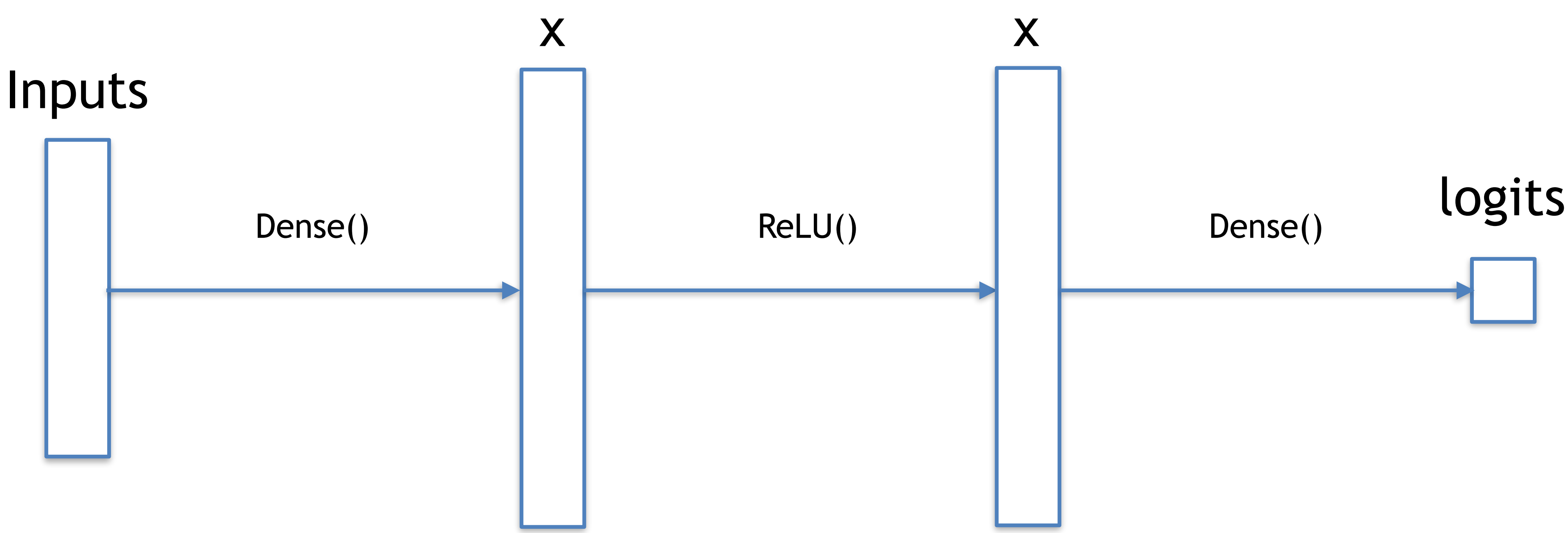
```
tf.reset_default_graph()
self.inputs = tf.placeholder(tf.float32, shape=[None, input_size])
self.labels = tf.placeholder(tf.float32, shape=[None])
self.learning_rate = tf.placeholder(tf.float32)
self.learning_rate = tf.placeholder(tf.float32)
```

shape=[None, input_size] -> [variable batch-size x input_size]

1-2 Add Layers

- Create input node -> **Create graph (Create output node)** -> Create session -> Run specific output node

```
with tf.variable_scope('model'):
    # TODO: Checkpoint 2
    # 1. Add additional Dense layers(>1) and ReLU
    x = Dense(units=16)(self.inputs)
    x = ReLU()(x)
    x = Dense(units=1)(x)
    logits = tf.reshape(x, [-1])
```



1-2 Add Layers

- Create input node -> **Create graph (Create output node)** -> Create session -> Run specific output node

```
with tf.variable_scope('model'):
    ...
    cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(
    labels=self.labels, logits=logits)
    self.loss = tf.reduce_mean(cross_entropy)
```

- Cross Entropy

  Let p denoted as the survival probability output from model, and q denoted as the true probability (0 or 1)

$$loss = \frac{1}{N} \sum_{i}^{N} q_i \cdot log(p_i) + (1 - q_i) \cdot log(1 - p_i)$$

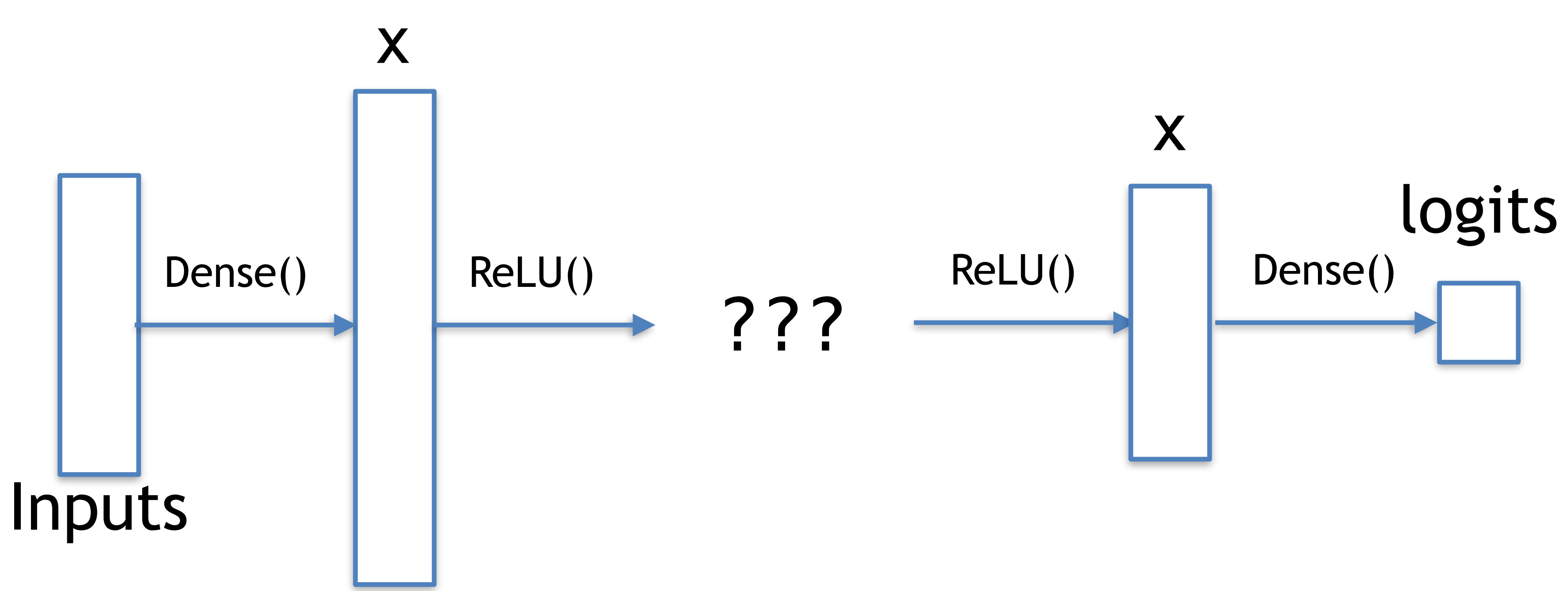
1-2 Add Layers

- Create input node -> **Create graph (Create output node)** -> Create session -> Run specific output node

```
with tf.variable_scope('model'):
    # TODO: Checkpoint 2
    # 1. Add additional Dense layers(>1) and ReLU
    x = Dense(units=16)(self.inputs)
    x = ReLU()(x)
    # x = Dense(???)
    # ...
    x = Dense(units=1)(x)
    logits = tf.reshape(x, [-1])
```

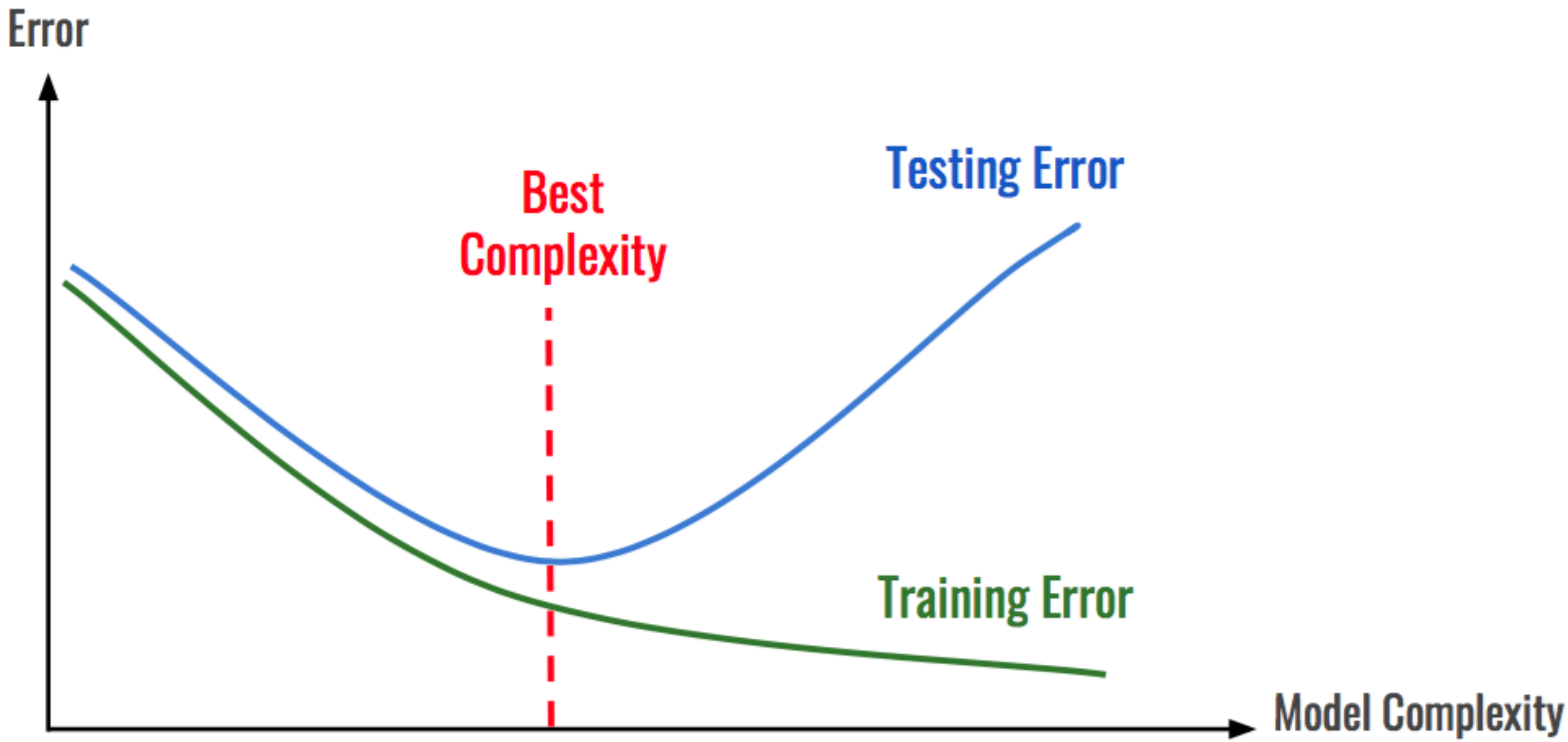
Checkpoint 2

1. Add additional Dense layers(>1) and ReLU
2. Increase epochs
3. Run LAB2.py
4. Show final progress bar



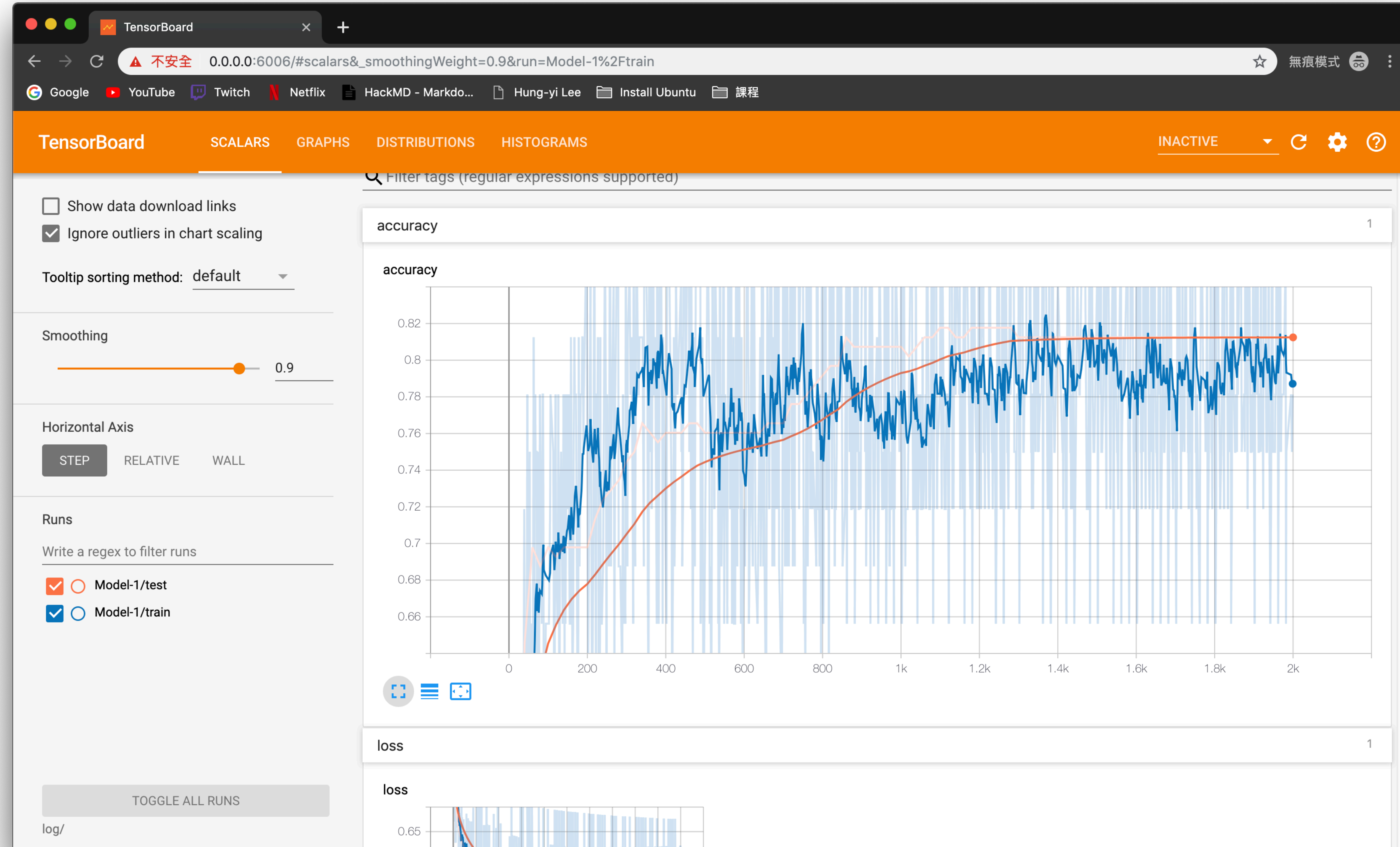
1-3 TensorBoard-Visualization

Visualizing the training process can help you to find out some characteristics of your model. For example, if over fitting or under fitting occurred, the loss curve of test data will respectively lie above and below the loss curve of train data.

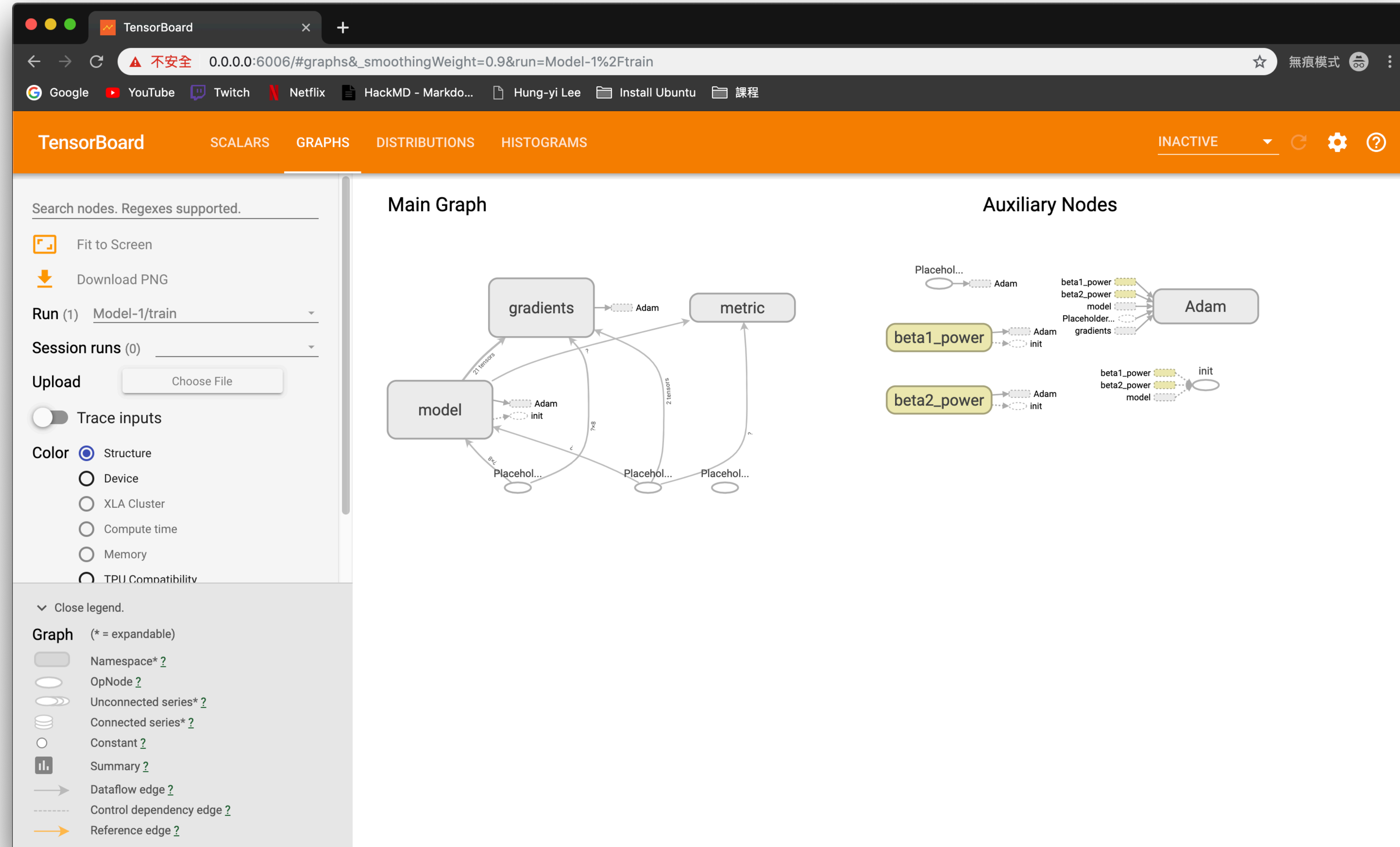


source: https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

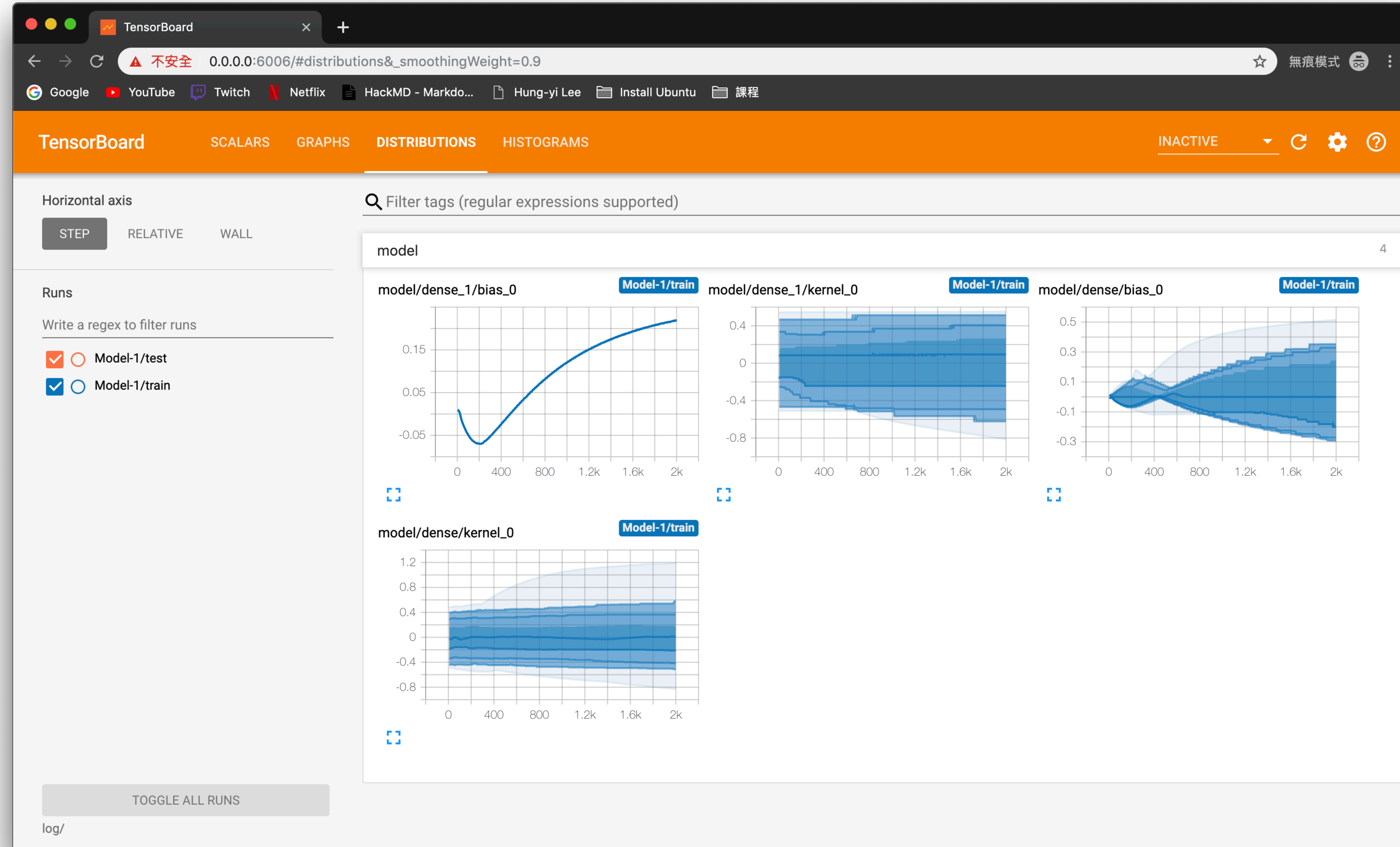
1-3 TensorBoard-Visualization



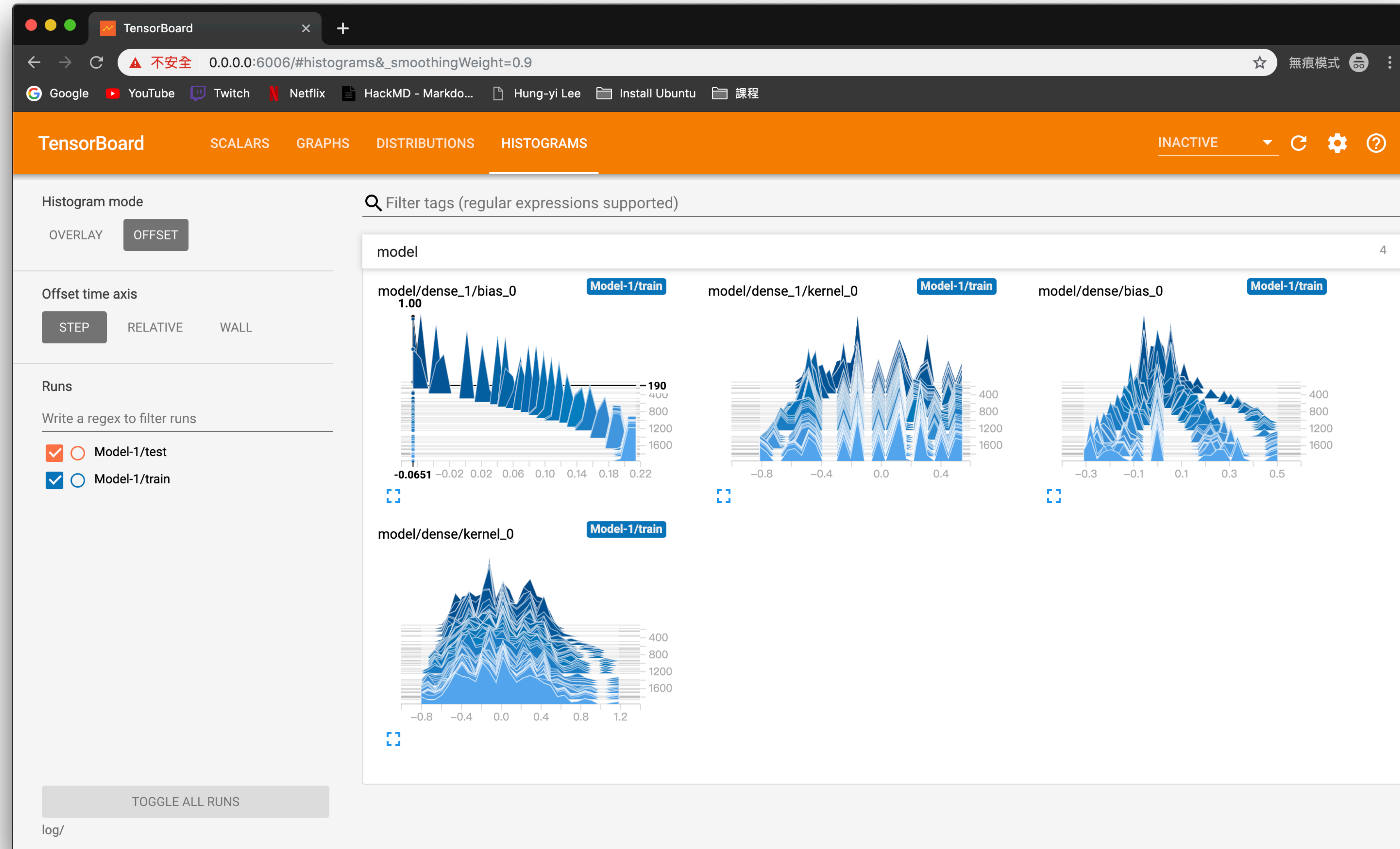
1-3 TensorBoard-Visualization



1-3 TensorBoard-Visualization



1-3 TensorBoard-Visualization



1-3 TensorBoard-Visualization

- Visualize variables

```
for variable in tf.trainable_variables(scope='model'):
    tf.summary.histogram(variable.name, variable)
tf.summary.scalar('loss', self.loss)
tf.summary.scalar('accuracy', self.accuracy)
self.merged = tf.summary.merge_all()
```

Checkpoint 3:
Show your model in TensorBoard

```
(venv) $ tensorboard --logdir=log/
```

1-4 Run the operation

- Create input node -> Create graph (Create output node) -> **Create session** -> Run specific output node

```
model = DNN(input_size=X.shape[1])
with tf.Session() as sess, tqdm(total=epochs) as pbar:
    sess.run(tf.global_variables_initializer())
    ...
```

```
tf.Session.__init__(target='', graph=None, config=None)
```

Args:

- **`target`**: (Optional.) The execution engine to connect to. Defaults to using an in-process engine. See Distributed TensorFlow for more examples.
- **`graph`**: (Optional.) The `Graph` to be launched (described above).
- **`config`**: (Optional.) A `ConfigProto` protocol buffer with configuration options for the session.

1-4 Run the operation

- Create input node -> Create graph (Create output node) -> Create session -> **Run specific output node**

```
model = DNN(input_size=X.shape[1])
with tf.Session() as sess, tqdm(total=epochs) as pbar:
    sess.run(tf.global_variables_initializer())
    ...
    for batch_x, batch_y in batch_generator(train_X, train_y, batch_size):
        feed = {
            model.inputs: batch_x,
            model.labels: batch_y,
            model.learning_rate: learning_rate_value,
        }
        train_loss, train_acc, _ = sess.run(
            [model.loss, model.accuracy, model.train_op], feed_dict=feed)
```

`fetches`: A single graph element or a list of graph elements
`feed_dict`: A dictionary that maps graph elements to values

1-4 Run the operation

- Create input node -> Create graph (Create output node) -> Create session -> **Run specific output node**

```
# TODO: Checkpoint 4
# 1. Compute loss and accuracy of test data
test_loss_sum = []
test_acc_sum = []
for batch_x, batch_y in batch_generator(test_X, test_y, batch_size):
    feed = {
        model.inputs: ???,
        model.labels: ???,
    }
    test_loss, test_acc = sess.run([???], feed_dict=???)
```

Checkpoint 4

Show the test_loss and test_acc in terminal

```
Epoch [199/200] test_loss=0.4231, test_acc=0.8368
Epoch [200/200] test_loss=0.4232, test_acc=0.8368
Epoch [200/200]: 100%|██████████| 200/200 [00:04<00:00, 42.50it/s, train_acc=0.8750, train_loss=0.3475]
```