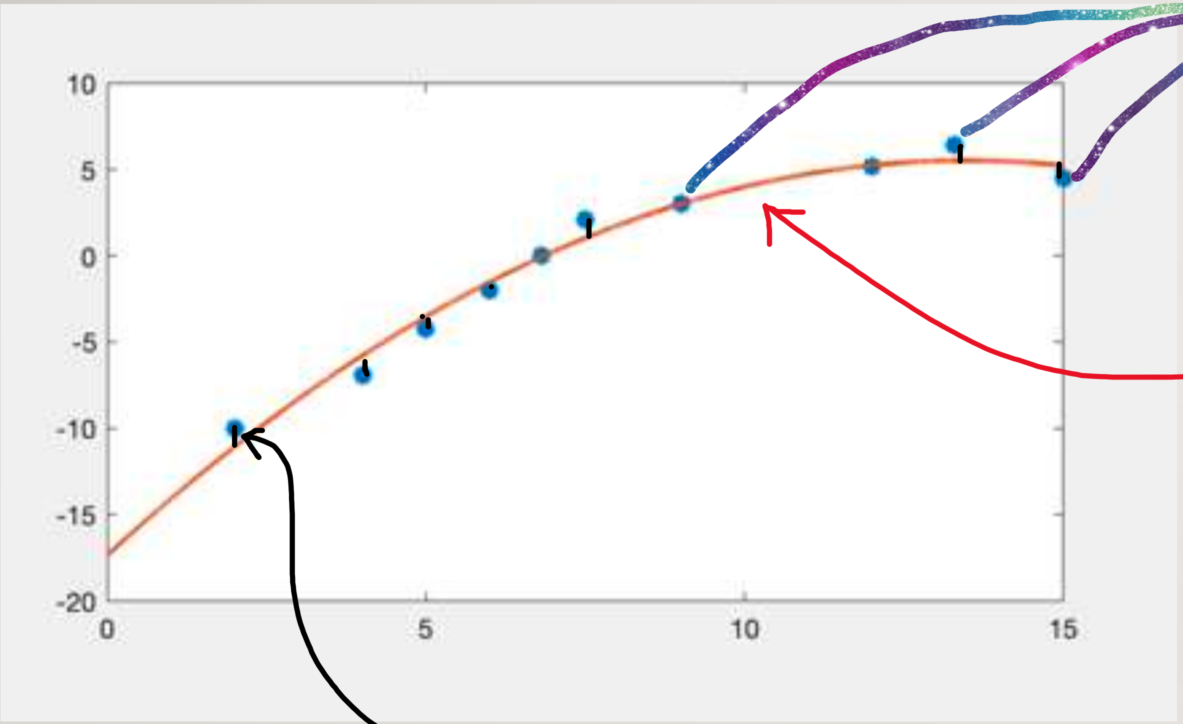
TENSORFLOW FOR DEEP LEARNING

SHUYUE JIA

https://github.com/SuperBruceJia

MANY OF YOU ARE AHEAD OF ME IN ACADEMIA SO I PROBABLY NEED MORE OF YOUR HELP THAN YOU DO MINE

Model Establishment for Supervised Learning



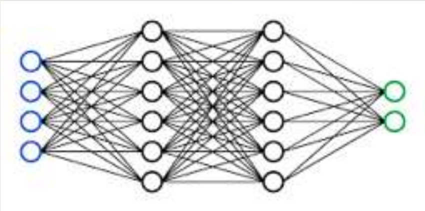
Ideal Model
g(x)

Loss Function = | g(x) - f(x) |

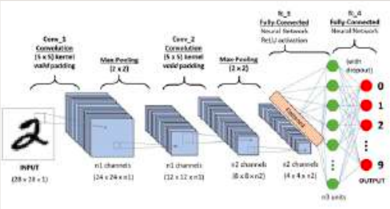
Established Model
f(x)

Loss

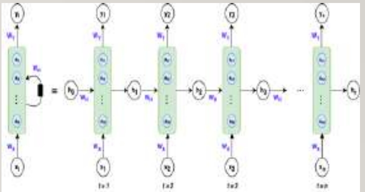
We want to minimize the loss



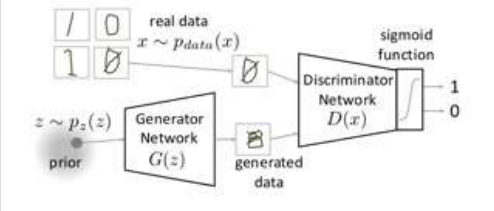
MLPs



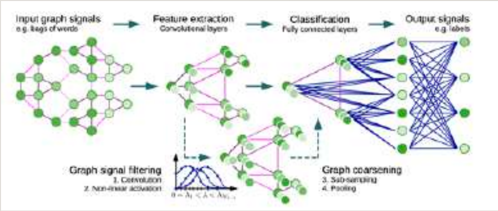
CNNs



RNNs



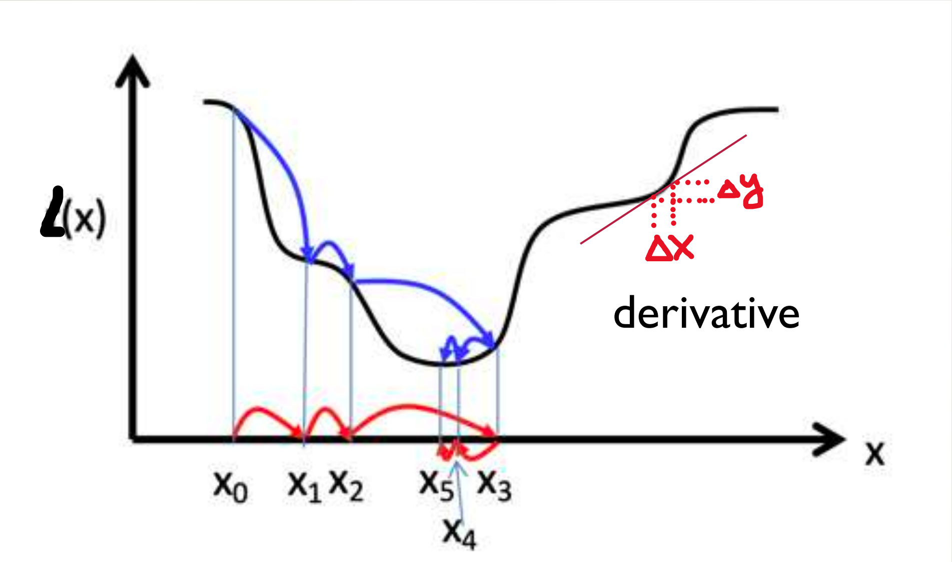
GANs



GCNs

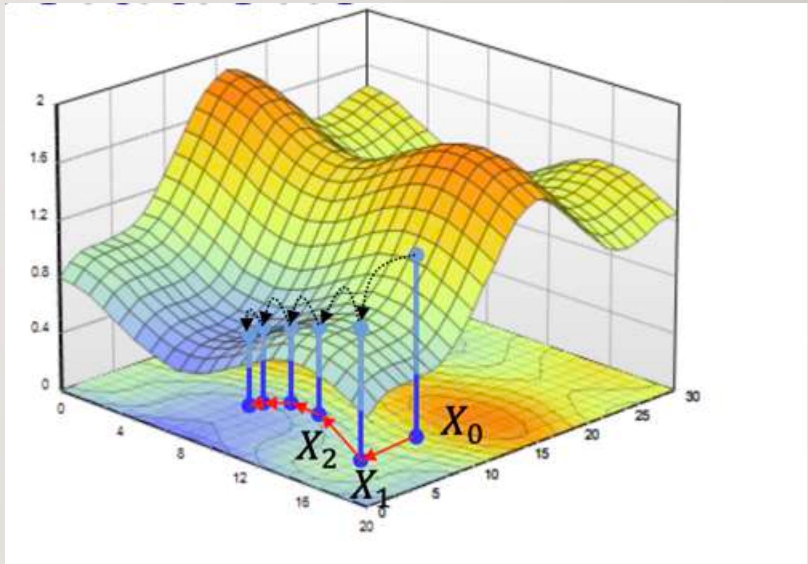
Gradient Descent Algorithm

Loss Function = | g(x) - f(x) $|_{minimize}$

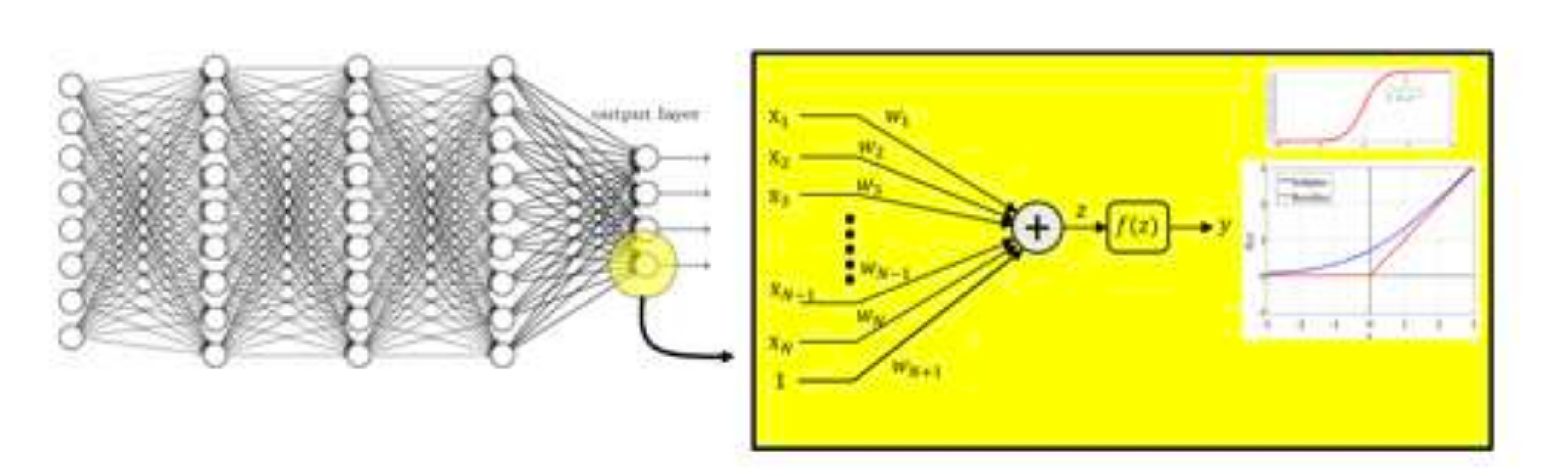


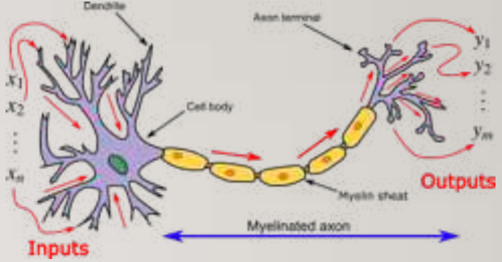
$$\alpha = \frac{\partial(|g(x) - f(x)|)}{\partial x}$$

$$x^{k+1} = x^k - \eta\alpha$$



Gradient Descent Algorithm for Neural Networks

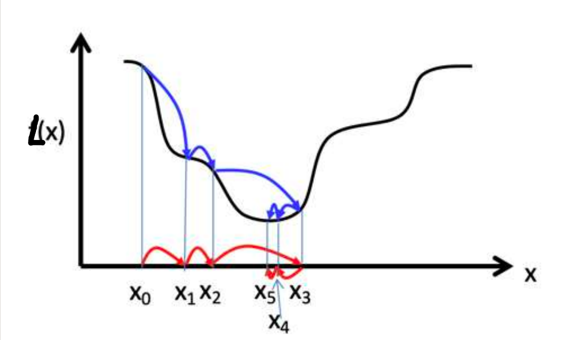




$$y = \sum_i w_i x_i + b$$

$$f(y) = \frac{1}{1+e^{-y}}$$

The parameters that we are training are W (weights) and b (biases).



$$x^{k+1} = x^k - \eta\alpha$$

$$\alpha = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial x}$$

$$y = \sum_i w_i x_i + b$$

$$f(y) = \frac{1}{1+e^{-y}}$$

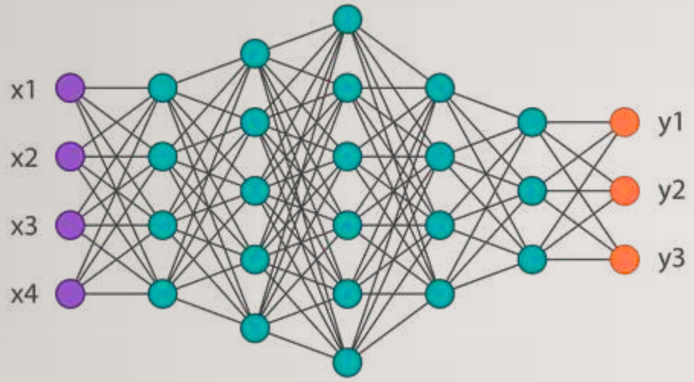
$$dw = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial w}$$

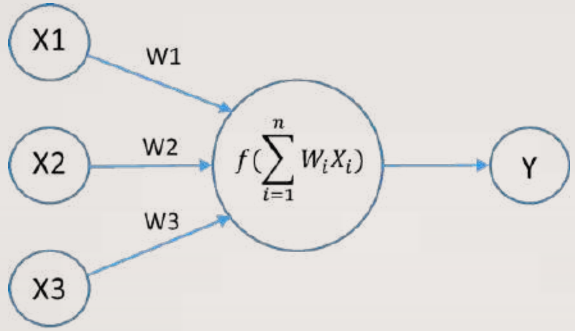
$$db = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial b}$$

$$w^{k+1} = w^k - \eta dw$$

$$b^{k+1} = b^k - \eta db$$

Back-propagation Algorithm (Chain Rule) for Neural Networks





$$y = \sum_i w_i x_i + b$$

$$f(y) = \frac{1}{1+e^{-y}}$$

$$dw = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial w}$$

$$db = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial b}$$

$$L = \frac{1}{2}(g(x) - f(x))^2$$

$$y = \sum_i w_i x_i + b$$

$$f(y) = \frac{1}{1+e^{-y}}$$

$$dw = \frac{\partial L}{\partial f(y)} \times \frac{\partial f(y)}{\partial y} \times \frac{\partial y}{\partial w}$$

$$= [g(x) - f(y)] \times [f(y) \times (1 - f(y))] \times X$$

TENSORFLOW: WHAT AND WHY?

- Open source software library for numerical computation of training neural networks (Deep Learning) using data flow graphs
- TensorFlow (Google), PyTorch (Facebook), MXNet (Microsoft)
- Flexibility + Scalability
- Popularity

FIRSTLY, INSTALL TENSORFLOW

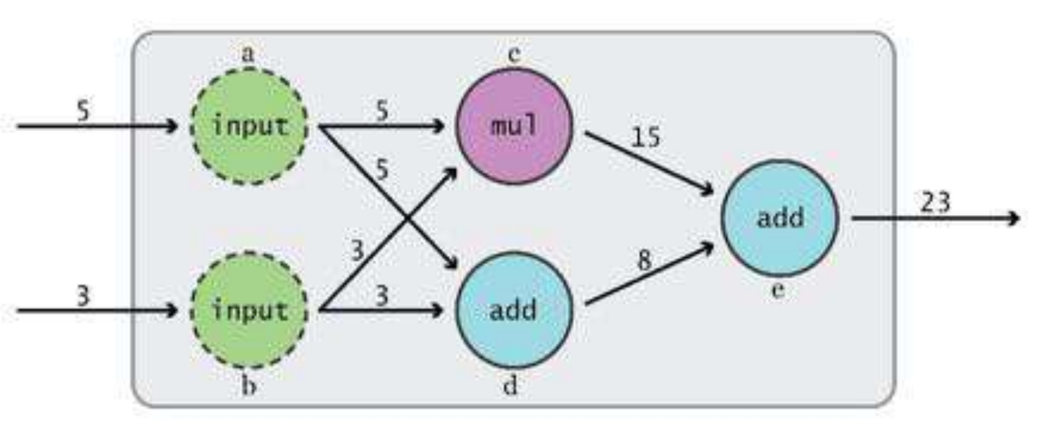
- conda create --name tensorflow python=3.7 numpy scipy
- conda activate tensorflow
- pip install tensorflow-gpu==1.14.0
- Recommended Python Package:
- numpy (Data Manipulation)， pandas (Data Analyze)， scipy (Scientific Computation)
- matplotlib and seaborn (Drawing Figures)
- scikit-learn (Machine Learning)

```
import tensorflow as tf
```

GRAPHS AND SESSIONS

DATA FLOW GRAPHS

- Phase 1: assemble a graph
- Phase 2: use a session to execute operations in the graph.

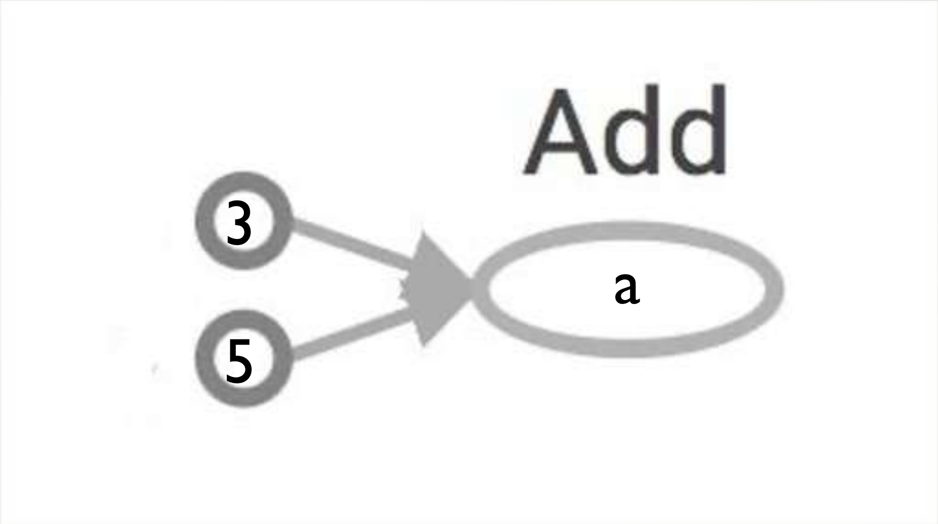


Nodes: operators, variables, and constants
Edges: tensors

Tensors are data.
TensorFlow = tensor + flow = data + flow

DATA FLOW GRAPHS

```
import tensorflow as tf
a = tf.add(3, 5)
print(a)
```



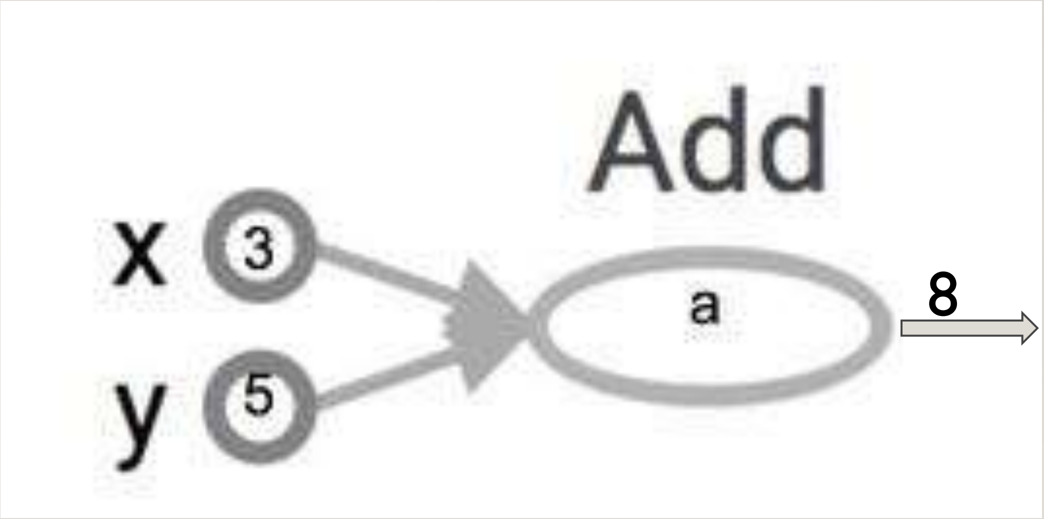

```
>> Tensor("Add:0", shape=(), dtype=int32)
```

(Not 8)

HOW TO GET THE VALUE OF A?

- Create a **session**, assign it to variable sess so we can call it later
- Within the session, evaluate the graph to fetch the value of a

```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
print(sess.run(a))      >> 8
sess.close()
```



1. Set up

```
# -*- coding: utf-8 -*-
# Hide the Configuration and Warning
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = '3'

# Import the Used Packages
import pandas as pd
import numpy as np
import tensorflow as tf
import random

# Clear the Stack
tf.reset_default_graph()
```

Training Set
The data in each row is a sample

	A	B	C	D	E	F	G	H
1	0.2372	20.025	0.3764	2.1013	0.5093	19.985	24.171	28.265
2	0.212	22.565	0.3469	2.2506	0.4827	25.451	26.75	37.381
3	0.2257	21.5	0.3665	2.1906	0.5013	22.191	25.926	30.769
4	0.2326	21.6	0.3563	1.6426	0.4759	21.038	26.155	29.859
5	0.2219	22.327	0.3568	2.0485	0.487	23.096	27.017	31.314
6	0.8398	1.6629	0.7627	1.0149	0.7404	2.5631	3.8825	2.2916
7	0.2147	21.888	0.3571	2.2763	0.49	23.487	26.58	33.482
8	0.4368	7.0138	0.4137	1.0933	0.4296	11.228	17.911	13.444
9	0.2449	20.812	0.3717	1.8044	0.5041	20.823	22.926	24.154
10	0.2454	20.061	0.3838	2.0129	0.5244	20.325	21.665	24.7
11	0.188	25.573	0.3344	2.7091	0.4729	29.258	29.299	42.045
12	0.2352	20.735	0.3705	2.0397	0.5003	20.709	24.759	29.256
13	0.238	19.386	0.3632	1.6607	0.4879	18.627	23.465	27.541
14	0.2302	19.547	0.3586	1.7941	0.4869	19.175	23.573	30.018
15	0.2188	19.874	0.3528	1.9407	0.4862	20.384	23.502	33.319
16	0.2057	23.402	0.3355	1.946	0.462	25.138	27.915	38.507
17	0.2228	21.516	0.3505	1.8321	0.4783	22.742	25.487	31.983
18	0.2221	20.387	0.365	2.1787	0.5006	21.274	24.692	30.558
19	0.2328	18.202	0.3724	1.9627	0.5067	18.202	21.747	29.854
20	0.201	20.624	0.3421	2.4204	0.4784	23.448	25.044	41.804
21	0.2252	18.438	0.3644	2.0561	0.4944	18.758	22.164	32.417
22	0.2039	23.166	0.3442	2.3844	0.4802	27.522	27.208	38.726
23	0.2321	20.381	0.3689	2.0549	0.5009	20.349	24.468	30.204
24	0.9997	0.9969	0.9952	0.9989	0.9928	0.9952	1.0202	0.9994
25	0.2278	21.095	0.3702	2.2041	0.5061	21.916	25.582	30.275
26	0.2201	20.901	0.3624	2.3663	0.4951	22.661	24.721	35.252
27	0.2292	20.929	0.37	2.1833	0.5026	21.765	24.963	30.558
28	0.2364	18.883	0.3724	1.8717	0.5082	18.704	22.102	27.799
29	0.283	16.623	0.3673	1.1996	0.4479	15.571	25.425	21.707
30	0.2206	21.699	0.3507	1.8797	0.4801	21.371	25.513	30.835
31	0.2118	21.239	0.3545	2.263	0.4977	24.407	24.762	36.78
32	0.2267	21.249	0.3603	1.9852	0.4895	21.243	25.761	30.203
33	0.2292	20.494	0.3698	2.1416	0.505	20.668	24.341	31.357
34	0.2323	18.619	0.3712	2.0372	0.5078	18.728	22.259	30.292
35	0.2148	20.526	0.3478	1.9827	0.4732	21.694	24.78	34.611
36	0.2397	20.615	0.3599	1.5438	0.4766	20.038	24.585	27.297
37	0.2232	22.592	0.3538	1.8945	0.4802	23.24	27.444	30.921
38	0.2005	20.504	0.3426	2.4367	0.4787	23.56	24.924	42.058

training_set

Training Labels

	A	B
1	1	
2	2	
3	1	
4	2	
5	1	
6	0	
7	1	
8	1	
9	0	
10	0	
11	1	
12	2	
13	0	
14	0	
15	0	
16	2	
17	0	
18	1	
19	0	
20	2	
21	0	
22	2	
23	2	
24	1	
25	1	
26	2	
27	2	
28	0	
29	0	
30	1	
31	0	
32	1	
33	2	
34	0	
35	0	
36	0	
37	1	
38	2	

training_label

2. Read the Dataset

```
# Read Training Data
train_data = pd.read_csv('training_set.csv', header=None)
train_data = np.array(train_data).astype('float32')

# Read Training Labels
train_labels = pd.read_csv('training_label.csv', header=None)
train_labels = np.array(train_labels).astype('float32')
train_labels = tf.one_hot(indices=train_labels, depth=4)
train_labels = tf.squeeze(train_labels).eval(session=sess)

# Read Testing Data
test_data = pd.read_csv('test_set.csv', header=None)
test_data = np.array(test_data).astype('float32')

# Read Testing Labels
test_labels = pd.read_csv('test_label.csv', header=None)
test_labels = np.array(test_labels).astype('float32')
test_labels = tf.one_hot(indices=test_labels, depth=4)
test_labels = tf.squeeze(test_labels).eval(session=sess)
```

3. INITIATION

```
# 初始化权重Weights函数
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.01)
    return tf.Variable(initial)

# 初始化偏置Biases函数
def bias_variable(shape):
    initial = tf.constant(0.01, shape=shape)
    return tf.Variable(initial)

# 定义卷积网络 stride==1， padding=‘SAME’输出大小等于输入大小
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

# 定义池化为最大池化 kernel大小为2*2， stride==1 ，padding=‘SAME’为尺寸减小一半
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

4. Design a graph

```
x_Reshape = tf.reshape(tensor=x, shape=[-1, 32, 20, 1])

# First Convolutional Layer
W_conv1 = weight_variable([3, 3, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_ Reshape, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# Second Convolutional Layer
W_conv2 = weight_variable([3, 3, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# First Fully Connected Layer
W_fc1 = weight_variable([5 * 8 * 64, 128])
b_fc1 = bias_variable([128])
h_pool2_flat = tf.reshape(h_pool2, [-1, 5 * 8 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Second Fully Connected Layer
W_fc2 = weight_variable([128, 64])
b_fc2 = bias_variable([64])
h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

# Output Layer: Thrid Fully Connected Layer
W_fc3 = weight_variable([64, 4])
b_fc3 = bias_variable([4])
prediction = tf.nn.softmax(tf.matmul(h_fc2_drop, W_fc3) + b_fc3)
```

5. LOSS FUNCTION, OPTIMIZER, AND ACCURACY

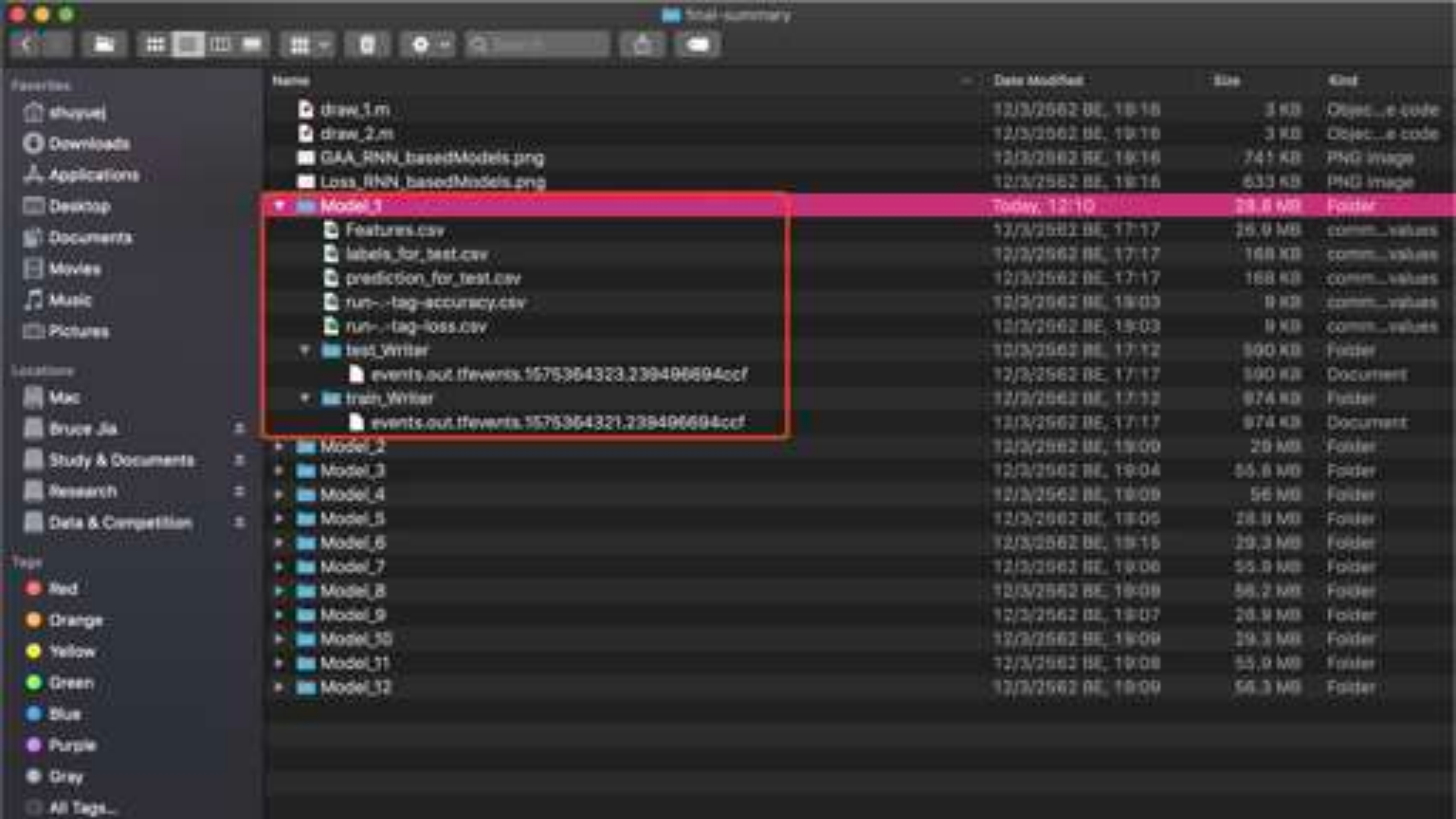
```
# Define Loss Function
loss = tf.reduce_mean(tf.square(y - prediction))

# Define Training Optimizer
train_step = tf.train.AdamOptimizer(1e-5).minimize(loss)

# Calculate Accuracy
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

6. USE THE SUMMARY TO SAVE ALL THE PARAMETERS THAT YOU WANT

```
tf.summary.scalar('loss', loss)


# Merge all the summaries
merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter(SAVE + '/train_Writer', sess.graph)
test_writer = tf.summary.FileWriter(SAVE + '/test_Writer')
```

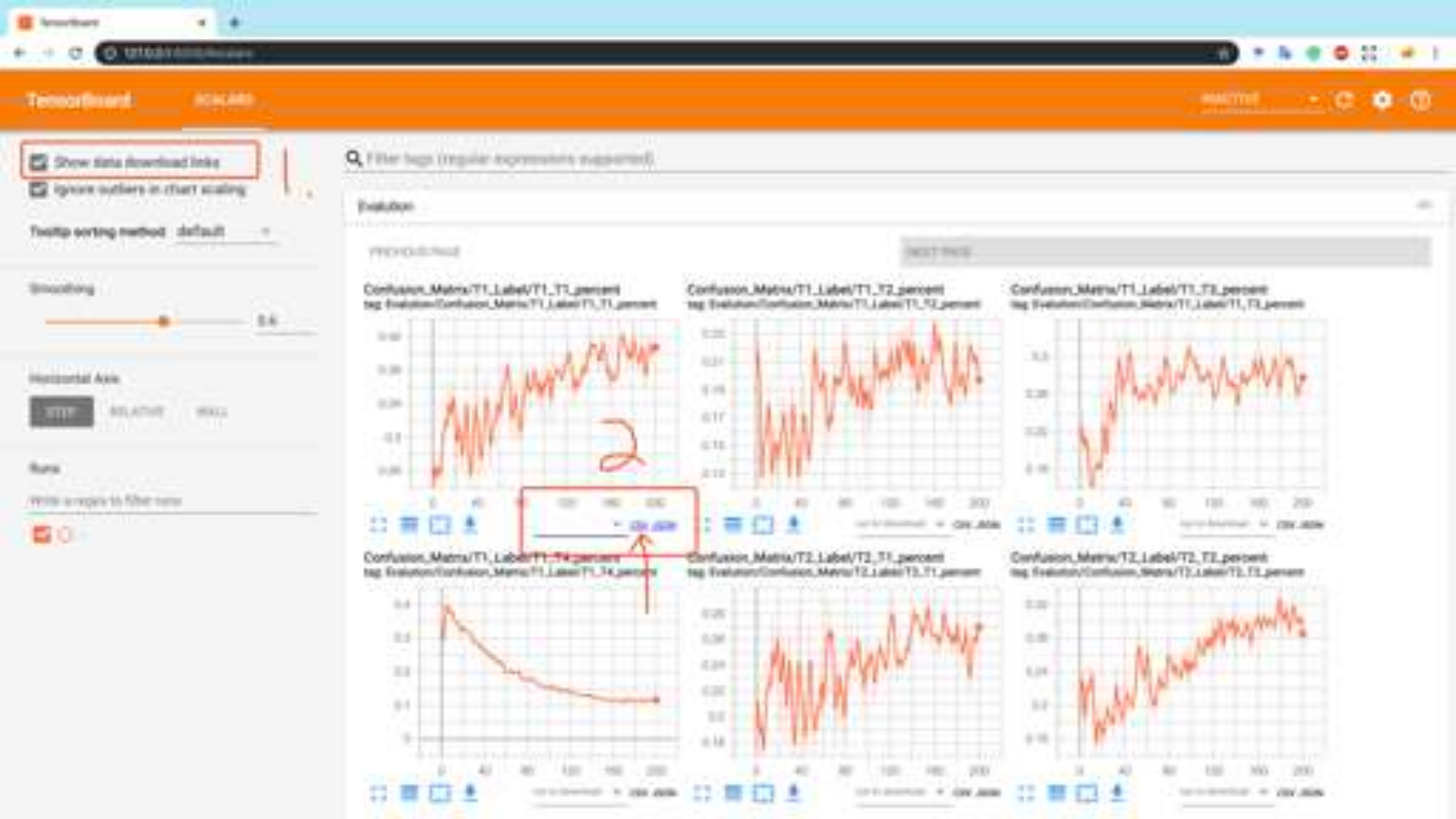
7. Use a session to execute

```
sess.run(tf.global_variables_initializer())
for epoch in range(num_epoch + 1):
    # Train the model
    for batch_index in range(n_batch):
        random_batch = random.sample(range(train_data.shape[0]), batch_size)
        batch_xs = train_data[random_batch]
        batch_ys = train_labels[random_batch]
        sess.run(train_step, feed_dict={x: batch_xs, y: batch_ys, keep_prob: keep_rate})

    # Show Accuracy and Loss on Training and Test Set
    train_accuracy, train_loss = sess.run([accuracy, loss], feed_dict={x: train_data, y: train_labels, keep_prob: 1.0})
    Test_summary, test_accuracy, test_loss = sess.run([merged, accuracy, loss], feed_dict={x: test_data, y: test_labels, keep_prob: 1.0}
    test_writer.add_summary(Test_summary, epoch)

    print("Iter " + str(epoch) + ", Testing Accuracy: " + str(test_accuracy) + ", Training Accuracy: " + str(train_accuracy))
    print("Iter " + str(epoch) + ", Testing Loss: " + str(test_loss) + ", Training Loss: " + str(train_loss))
    print('\n')
```


Name
Date Modified
Size
Kind
draw_1.m
draw_2.m
GAA_RNN_basedModels.png
Loss_RNN_basedModels.png
Model_1
Today, 12:10
Folder
Features.csv
labels_for_test.csv
prediction_for_test.csv
run-.-tag-accuracy.csv
run-.-tag-loss.csv
test_Writer
events.out.tfevents.1575364323.239496694ccf
train_Writer
events.out.tfevents.1575364321.239496694ccf
Model_2
Model_3
Model_4
Model_5
Model_6
Model_7
Model_8
Model_9
Model_10
Model_11
Model_12
Downloads
Applications
Desktop
Documents
Movies
Music
Pictures
Mac
Bruce Jia
Study & Documents
Research
Data & Competition
Red
Orange
Yellow
Green
Blue
Purple
Gray
All Tags...

8. 使用以下指令去可视化训练结果

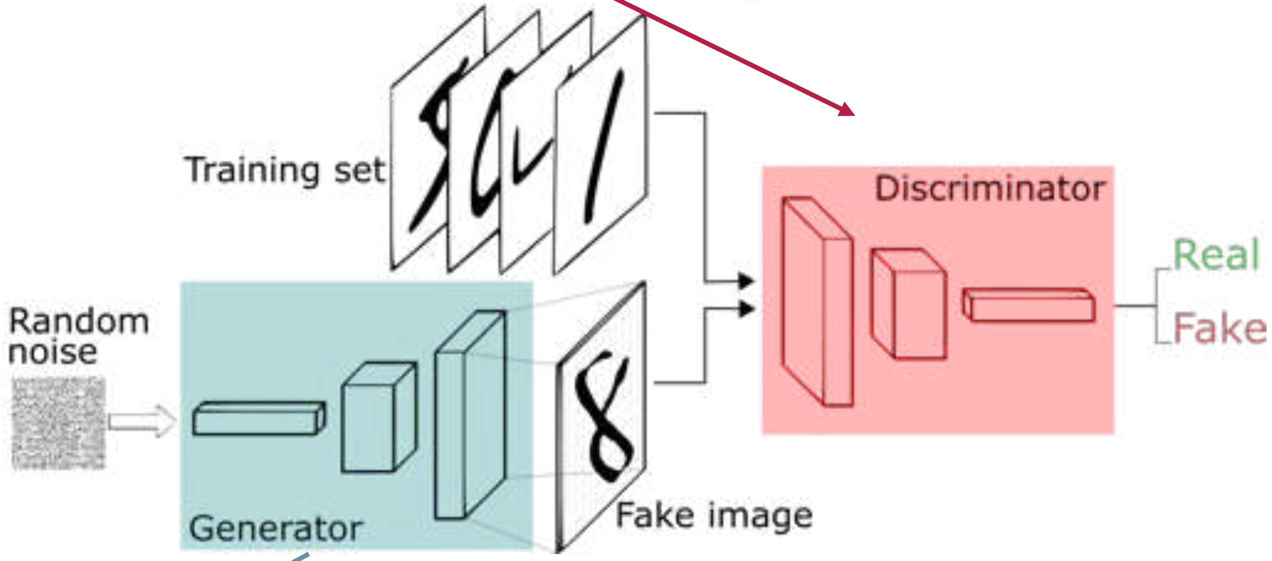
tensorboard --logdir=“路径绝对地址" --host=127.0.0.1


TensorBoard
SCALARS
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method default
Smoothing
Horizontal Axis
Runs
Evaluation
Confusion_Matrix/T1_Label/T1_T1_percent
Confusion_Matrix/T1_Label/T1_T2_percent
Confusion_Matrix/T1_Label/T1_T3_percent
Confusion_Matrix/T1_Label/T1_T4_percent
Confusion_Matrix/T2_Label/T2_T1_percent
Confusion_Matrix/T2_Label/T2_T2_percent
2

GENERATIVE ADVERSARIAL NETWORKS - GANS

We want to minimize the Discriminator Loss

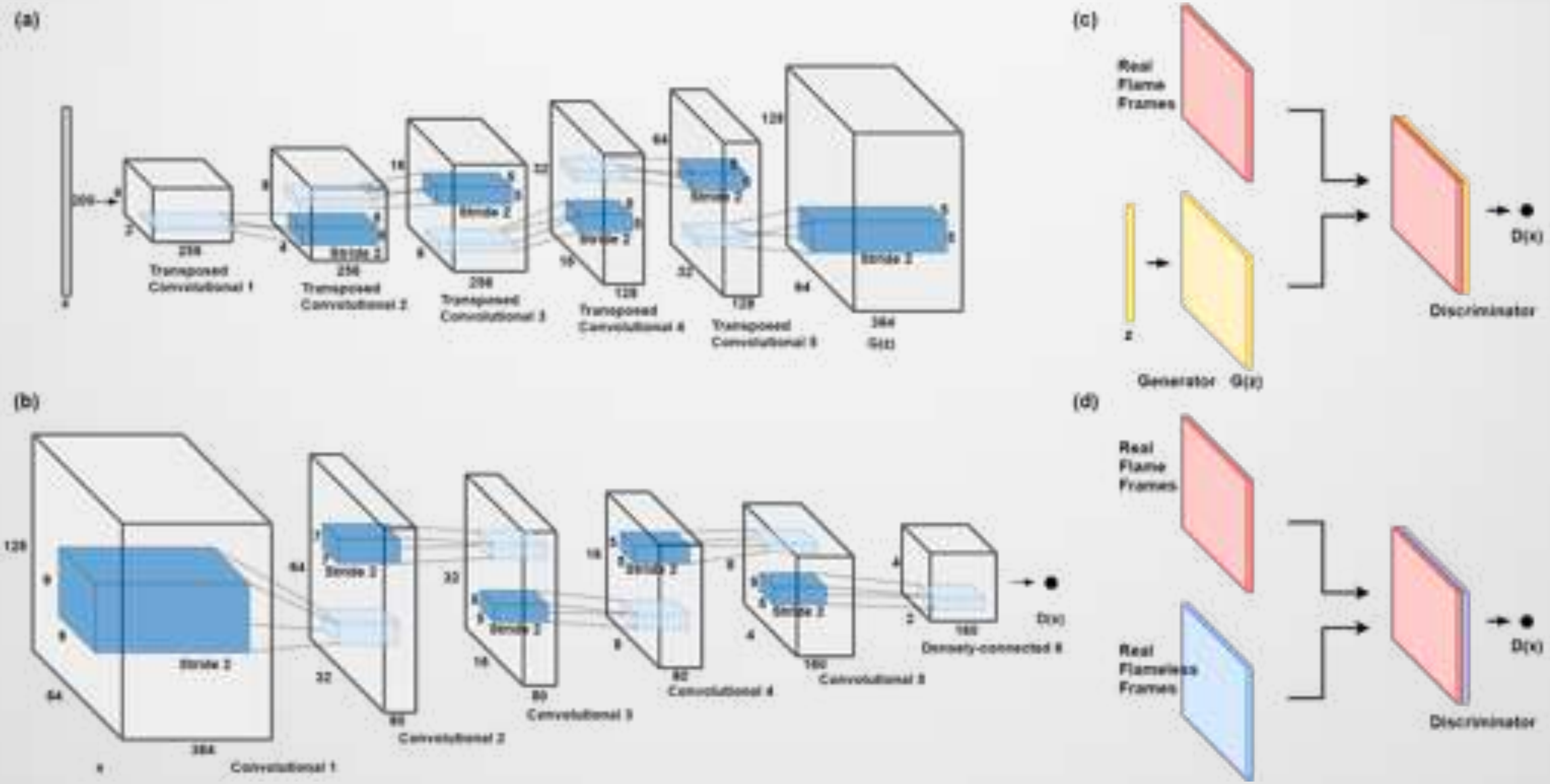
$$
\begin{aligned}
E(G, D) &= \frac{1}{2}\mathbb{E}_{x\sim p_t}[1 - D(x)] + \frac{1}{2}\mathbb{E}_{z\sim p_z}[D(G(z))] \\
&= \frac{1}{2}\left(\mathbb{E}_{x\sim p_t}[1 - D(x)] + \mathbb{E}_{x\sim p_g}[D(x)]\right)
\end{aligned}
$$



We want to maximize the Discriminator Loss

$$\max_G \left(\min_D E(G, D)\right)$$

The Architecture of the DCGANs



https://github.com/znxlwm/tensorflow-MNIST-GAN-DCGAN
https://github.com/sheqi/GAN_Review
https://sthalles.github.io/intro-to-gans/

THANKS

https://github.com/SuperBruceJia/EEG-Motor-Imagery-Classification-CNNs-TensorFlow