SVHN_RNN

本项目的英文 ReadMe 文档及代码参见 Github：
https://github.com/lixuan0023/SVHN_RNN

Contents

模型概述

介绍

这个项目是使用 Tensorflow 完成优达学城(Udacity.com)的一个练习，参考了 SVHNClassifier 和 im2txt。SVHN_RNN 模型结合了 CNN 与 RNN 两种神经网络，其中 CNN 层将输入图像编码成一个固定长度的特征向量，而 RNN 层将特征解码成了一串数字（门牌号码）。

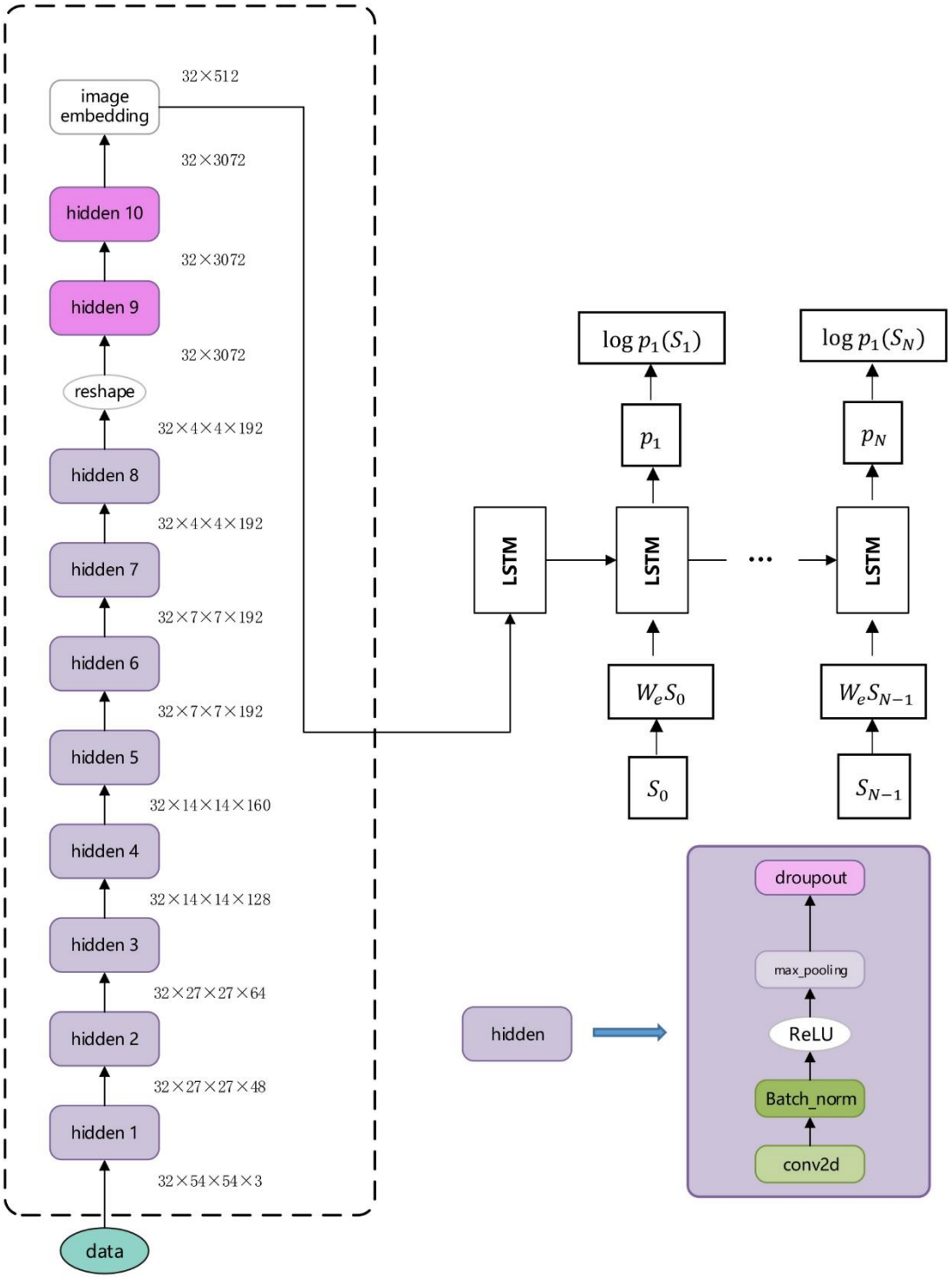
模型结构

SVHN_RNN 模型是一个编码-解码式（*encoder-decoder*）的神经网络示例。

图像编码器是使用的卷积神经网络，这种网络广泛用于图像任务，本模型参考 SVHNClassifier。

而解码器是使用于 Long Short-term Memory (LSTM) 网络。这种网络通常用于序列模型任务，如语言模型，机器翻译。在 SVHN_RNN 中，LSTM 网络用于将图像信息解码成数字序列。

门牌上的数字被表示成嵌入(embedding)模型。每个在字典中的(vocabulary)数字表示成一个定长的向量特征。网络结构如下图。



在图中，$\{\boldsymbol{S_0}, \boldsymbol{S_1} \ldots, \boldsymbol{S_{N-1}}\}$ 为门牌号数字，$\{\boldsymbol{W_e S_0}, \boldsymbol{W_e S_1} \ldots, \boldsymbol{W_e S_{N-1}}\}$是相应数据的特征向量(embedding vector)。LSTM 的输出$\{p_1, p_2, \ldots, p_N\}$是模型预测下一个数字出现的概率分布。{log $p_1(s_1)$, log $p_2(s_2)$, ..., log $p_N(s_N)$} 是每一步预测出数字的 log 似然概率。

卷积网络的隐藏层如图右下角。包括了卷积层, batch normalization, ReLU, max pooling 和 dropout。

准备阶段

- 关于硬件

  SVHN_RNN 模型的训练时间取决于使用的硬件性能，计算能力。以我的实验来说，是在一台拥有 8 片 *NVIDIA Tesla K20c* 的 GPU 上运行，整个过程大概需要 4-5 天时间来达到最高的训练精度。

- 环境配置

  TensorFlow 1.0 or greater (instructions)

  NumPy (instructions)

  h5py (instructions)

  pillow (instructions)

```
In Ubuntu:
$ sudo apt-get install libhdf5-dev
$ sudo pip install h5py

Install pillow:
$ sudo pip install Pillow
```

- 准备数据

  为了训练模型，需要先将训练数据转化成 TFRecord 格式。它是由包含 `tf.SequenceExample` 协议的数据组成。每个 `tf.SequenceExample` 又包括一个图像 (PNG 格式)，门牌号数字等。

  注：数字 0, 1,…,9 分别由对应的标签 ‘0’, ’1’, …, ‘9’表示。标签’10’代表开始标志，标签’11’代表结束标志。门牌数字的格式为 `10,XXX,11`.

  数据来自 Street View House Numbers (SVHN)，将下载好的数据解压成如下文件结构。

```
SVHN_RNN
    - data
        - extra
            - 1.png
            - 2.png
            - ...
            - digitStruct.mat
        - test
            - 1.png
            - 2.png
            - ...
            - digitStruct.mat
        - train
            - 1.png
            - 2.png
            - ...
            - digitStruct.mat
```

训练模型

1. Convert to TFRecords format

```
$ python convert_to_tfrecords.py --data_dir ./data
```

2. Train

```
$ python train.py --data_dir ./data --train_logdir ./logs/train
```

3. Retrain if you need

```
$ python train.py --data_dir ./data --train_logdir ./logs/train2 --restore_checkpoint ./logs/train/latest.ckpt
```

4. Evaluate

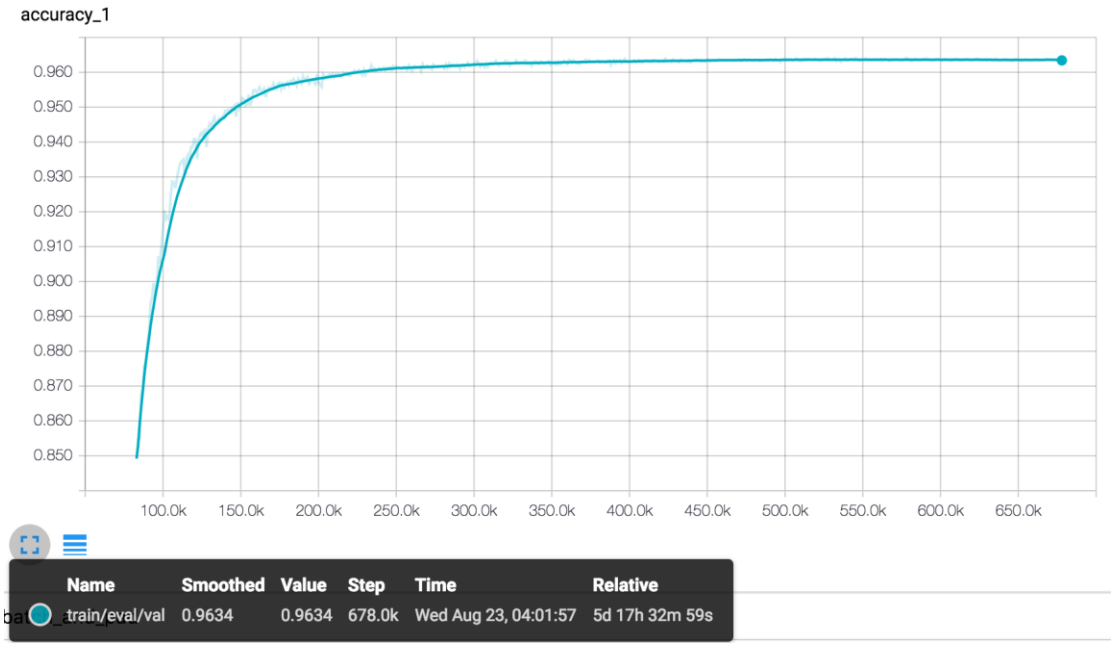
```
$ python eval.py --data_dir ./data --checkpoint_dir ./logs/train --eval_logdir ./logs/eval
```

5. Visualize

```
$ tensorboard --logdir ./logs
```

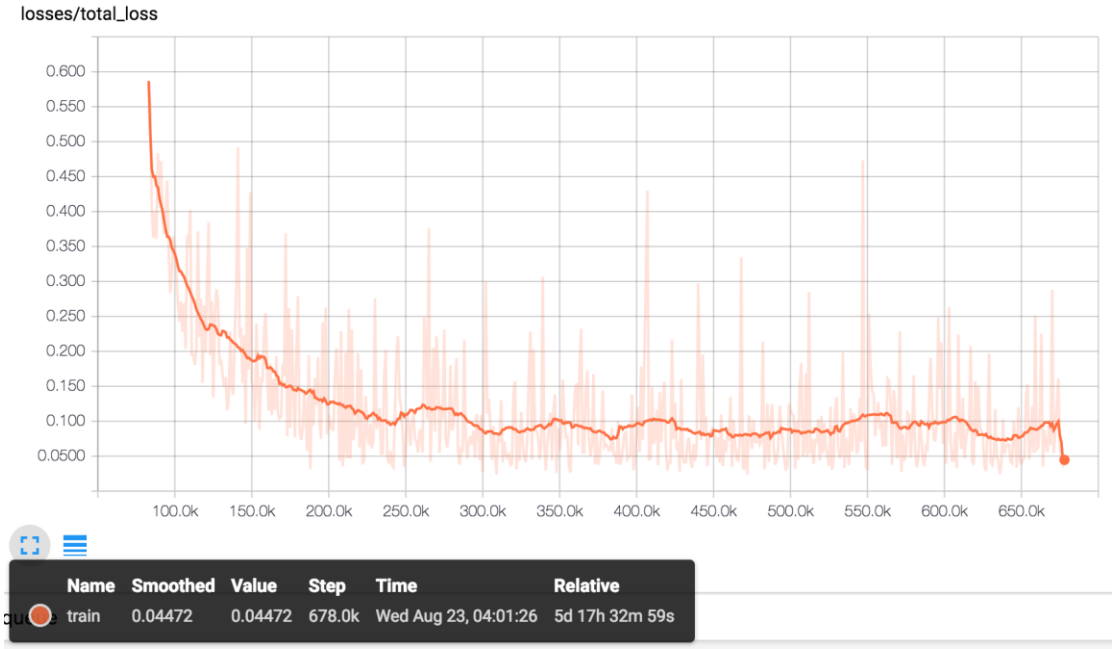
可视化(Tensorboard)

精度



在大概第 6 天的时候，测试集上的训练精度达到 96.34

Loss



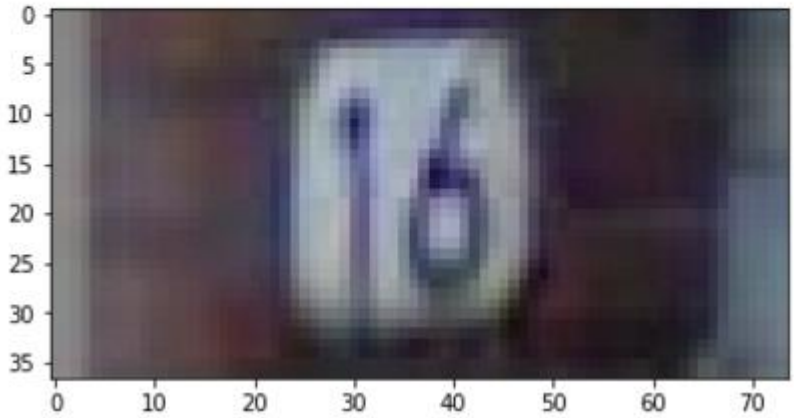
生成门牌号(Inference)

Run in Jupyter

```
Open 'run_inference.ipynb' in Jupyter
```

结果如下

```
The candidate result as follow.
No.     result  probability
 1       16      0.578
 2       18      0.388
 3       10      0.013
```



```
The candidate result as follow.
No.      result  probability
 1        257     0.981
 2        255     0.005
 3        254     0.004
```