摘要

本周周报的主要内容是如何运行使用magenta示例模型中的melody\_rnn，总结了**如何使用已训练模型**以及**自己训练新模型生成旋律**的过程，包括创建数据集，划分训练集和评估集，创建训练模型和评估模型以及最终的生成旋律等。另外进行了**tensorflow**的进一步学习，自己敲了一些简单的实现代码。其中关于使用melody\_rnn模型生成旋律的内容是我自己通过参考原项目的README.md（地址在下方），以及查找了一些资料，并加上自己的理解写出来的，不保证完全正确，有些音乐乐理方面的知识还不是很了解，我会在遇到的时候慢慢补习。<https://github.com/tensorflow/magenta/tree/master/magenta/models/melody_rnn>

# Melody RNN

这个模型主要使用LSTM（Long Short-Term Memory）将语言建模应用到旋律生成中来。使用基本的one-hot编码来表示提取的旋律作为LSTM的输入（one-hot编码方式介绍参考<http://blog.clzg.cn/blog-1579109-884831.html> 此处不过多赘述，这种编码方式主要就是使特征变得稀疏，使程序更好地处理输入的数据）。

运行环境配置：关于**magenta**安装

<https://github.com/tensorflow/magenta#installation>

在配置好环境后我们可以选择使用已经训练好的模型或自己训练模型来生成旋律；

## 使用已训练好的模型（Pre-trained）生成旋律（大致过程）：

**BUNDLE\_PATH=<absolute path of .mag file>** #声明包的绝对路径；

**CONFIG=<one of 'basic\_rnn', 'lookback\_rnn', or 'attention\_rnn', matching the bundle>**

#声明配置文件：此处三个文件均为已经训练好的模型的.mag包文件；

**melody\_rnn\_generate \**#生成旋律；

**--config=${CONFIG} \**#调用配置文件；

**--bundle\_file=${BUNDLE\_PATH} \**#调用包；

**--output\_dir=/tmp/melody\_rnn/generated \**#指明MIDI文件的输出路径；

**--num\_outputs=10 \**#定义产生旋律的数量；

**--num\_steps=128 \**#这个参数的原版解释是how long each melody will be in 16th steps (128 steps = 8 bars) 我的理解是每个旋律在16步中持续多长时间，也就是每小节该旋律的持续时间（128步等于8小节，因此每16步是1小节）；

**--primer\_melody="[60]"**# 由于模型在开始生成连续音符之前，至少需要**将一个音符馈送到模型**。我们可以使用--primer\_melody方法，用Python列表的字符串表示来指定一个启动旋律。列表中的值应该是遵循melodies\_lib.Melody格式的内容（其中-2 代表无事件，-1 代表音符关闭事件，值0到127代表该MIDI音高的音符事件）。例如--primer\_melody="[60, -2, 60, -2, 67, -2, 67, -2]"，使用Twinkle Twinkle Little Star的前四个音符（小星星用数字音符表示前四个音依次是1155）来填充该模型。除了使用--primer\_melody方法，我们可以使用--primer\_midi方法调用存储于MIDI文件中的旋律，来填充我们的模型。例如，--primer\_midi=<absolute path to magenta/models/shared/primer.mid>，将使用该MIDI文件中的旋律来填充该模型。如果既不使用--primer\_melody方法也不使用--primer\_midi方法，那么程序会在指定模型音符范围内的随机音符中，随机选取一个音符将作为启动音符，剩余的音符将由模型生成。--primer\_melody="[60]"代表运行本例后将会产生一段从middle C开始的旋律。

## 自己训练模型生成旋律（Train my own）：

下面分步骤对自己训练模型生成旋律的过程进行阐述：

**①创建音符序列（NoteSequences）**

我们的第一步需要将MIDI文件的集合转换为音符序列（音符序列属于协议缓冲区，它是一种快速有效的数据格式，并且比MIDI文件更易于使用）。不过首先我们需要创建自己的MIDI数据集（关于MIDI数据集可以使用

<http://colinraffel.com/projects/lmd/>中的**Lakh MIDI数据集**），之后再转换成音符序列：

**INPUT\_DIRECTORY=<folder containing MIDI and/or MusicXML files. can have child folders.>**  #定义数据集的输入路径

**SEQUENCES\_TFRECORD=/tmp/notesequences.tfrecord** #定义输出路径

**convert\_dir\_to\_note\_sequences \**  #转换数据集—>音符序列

**--input\_dir=$INPUT\_DIRECTORY \** #MIDI数据集输入路径

**--output\_file=$SEQUENCES\_TFRECORD \**#音符序列输出路径

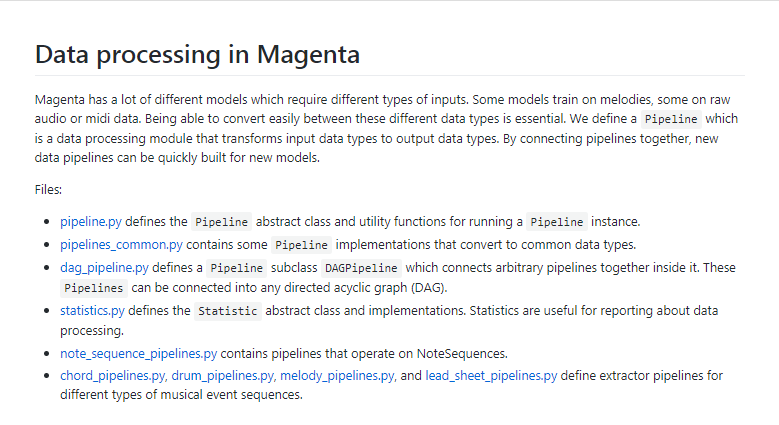
**--recursive** #递归

# 此处我们假设已输出了NoteSequences /tmp/notesequences**.tfrecord**文件，ps：TFRecord 文件包含NoteSequence协议缓冲区的文件；

另外在自己创建模型时需要调用很多magneta中用于数据处理的.py文件，具体的每个文件的内容及如何使用可以参考API中的内容:

**magenta的数据处理API地址**

<https://github.com/tensorflow/magenta/tree/master/magenta/pipelines>



**②创建序列样本（SequenceExamples）**

在训练和评估过程时，序列样本会被输入到模型中。每个序列样本包含一个输入序列和代表一个旋律的一系列标签。我们需要从上一部得到的NoteSequences中提取旋律，并将它们保存为SequenceExamples。此过程将生成两个SequenceExamples集合，一个用于训练，另一个用于评估，其中评估序列样本的如何划分由评估率**--eval\_ratio**确定。若评估率为0.10，则提取的旋律的10％将保存在评估集中，90％将被保存在训练集中。

**melody\_rnn\_create\_dataset \**

**--config=<one of 'basic\_rnn', 'lookback\_rnn', or 'attention\_rnn'> \**#调用配置文件

**--input=/tmp/notesequences.tfrecord \**#输入音符序列

**--output\_dir=/tmp/melody\_rnn/sequence\_examples \**#输出序列样本

**--eval\_ratio=0.10** #评估率

**③训练和评估模型**

首先是***训练模型***：

**melody\_rnn\_train \**

**--config=attention\_rnn \**

**--run\_dir=/tmp/melody\_rnn/logdir/run1 \**

#定义存储此次运行的检查点和TensorBoard数据的目录

**--sequence\_example\_file=/tmp/melody\_rnn/sequence\_examples/training\_melodies.tfrecord \**#定义将被输入进模型中的SequenceExamples的TFRecord文件路径

**--hparams="batch\_size=64,rnn\_layer\_sizes=[64,64]" \**

#--hparams（可选）用于指定超出默认值的超参数。在此示例中，指定自定义批量大小为64而不是默认批量大小为128。因为使用较小的批量大小可以帮助减少内存使用量，从而可以在训练较大型号时解决潜在的内存不足问题。我们还将使用每个64个单位的2层RNN，而不是默认的2层128个单位。这将使我们的模型训练**更快**。不过如果有足够的计算能力，则可以尝试使用较大的图层大小以获得更好的结果。

**--num\_training\_steps=20000**

#（可选）退出训练循环之前要执行的更新步骤。如果没有指定，训练循环将一直运行，直到手动终止。

对于***评估模型***：

评估工作可以并行进行。其中--run\_dir，--hparams以及--num\_training\_steps应取与训练（train）时相同的值。可以保证在不更新任何权重的情况下同时进行评估。

**melody\_rnn\_train \**

**--config=attention\_rnn \**

**--run\_dir=/tmp/melody\_rnn/logdir/run1 \**

**--sequence\_example\_file=/tmp/melody\_rnn/sequence\_examples/eval\_melodies.tfrecord \**#指向被分离出的单独一组评估旋律

**--hparams="batch\_size=64,rnn\_layer\_sizes=[64,64]" \**

**--num\_training\_steps=20000 \**

**--eval**

运行成功之后，可以运行TensorBoard来查看用于训练和评估的数据，**tensorboard --logdir=/tmp/melody\_rnn/logdir**。之后转到http：// localhost：6006查看TensorBoard即可。

**④生成旋律（Generate Melodies）**

旋律可以在训练期间或之后生成。运行下面的命令可以使用训练好的模型的最新检查点文件生成一组旋律：

**melody\_rnn\_generate \**

**--config=attention\_rnn \**

**--run\_dir=/tmp/melody\_rnn/logdir/run1 \**

#此处的目录应与训练时的目录相同，--run\_dir内部的子目录train是加载最新检查点文件的位置。如果我们使用-- /tmp/melody\_rnn/logdir/run1/作为run\_dir，那么tmp/melody\_rnn/logdir/run1/train就是最新的检查点文件所在的位置。

**--output\_dir=/tmp/melody\_rnn/generated \**

#--output\_dir是生成的MIDI文件被保存的文件路径；

**--num\_outputs=10 \**

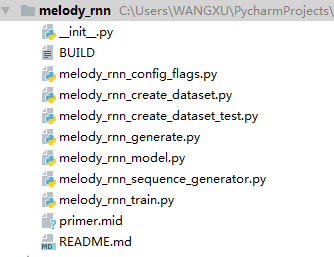
**--num\_steps=128 \**

**--hparams="batch\_size=64,rnn\_layer\_sizes=[64,64]" \**

#此处的—hparams与训练时的超参数相同，尽管其中的一些值将会被忽略；

**--primer\_melody="[60]"**#此处设定启动旋律的音符

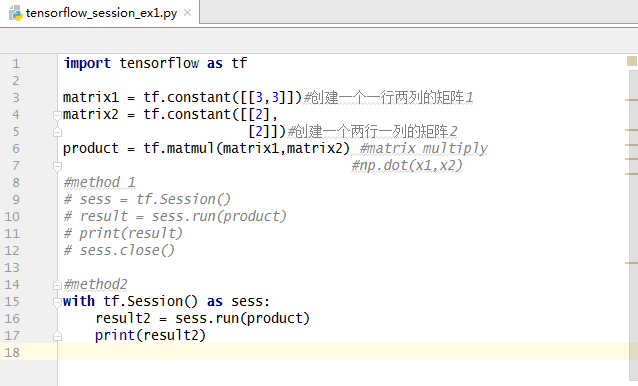
至此，自己训练的模型便可以生成旋律了，我们可以看到源程序的文件结构基本

了自己训练模型生成旋律的过程，包括创建数据集，将数据集转化为音符序列，训练模型等，关于每个文件的详细注释，我会在以后的报告中给出。

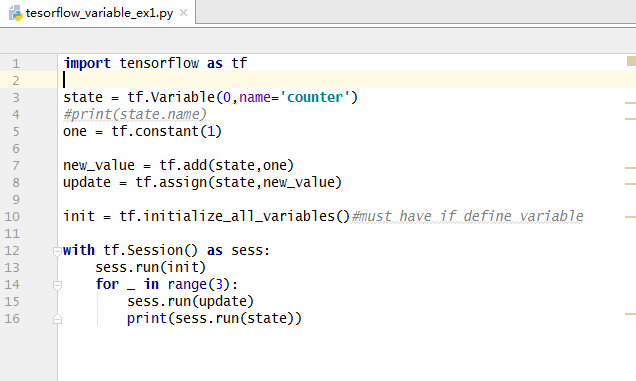
# Tensorflow学习内容

**①Session 会话控制**

此部分主要学到了run Session的两种方法（小例子计算矩阵相乘并打印结果）

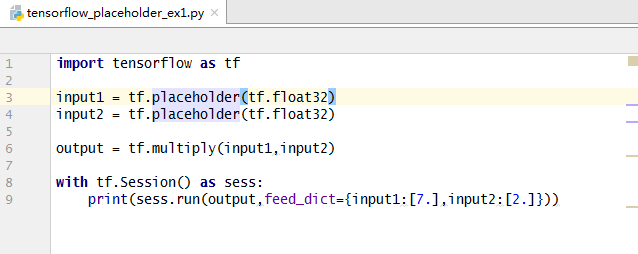


**②Variable** 主要讲到的变量的声明、初始化以及调用



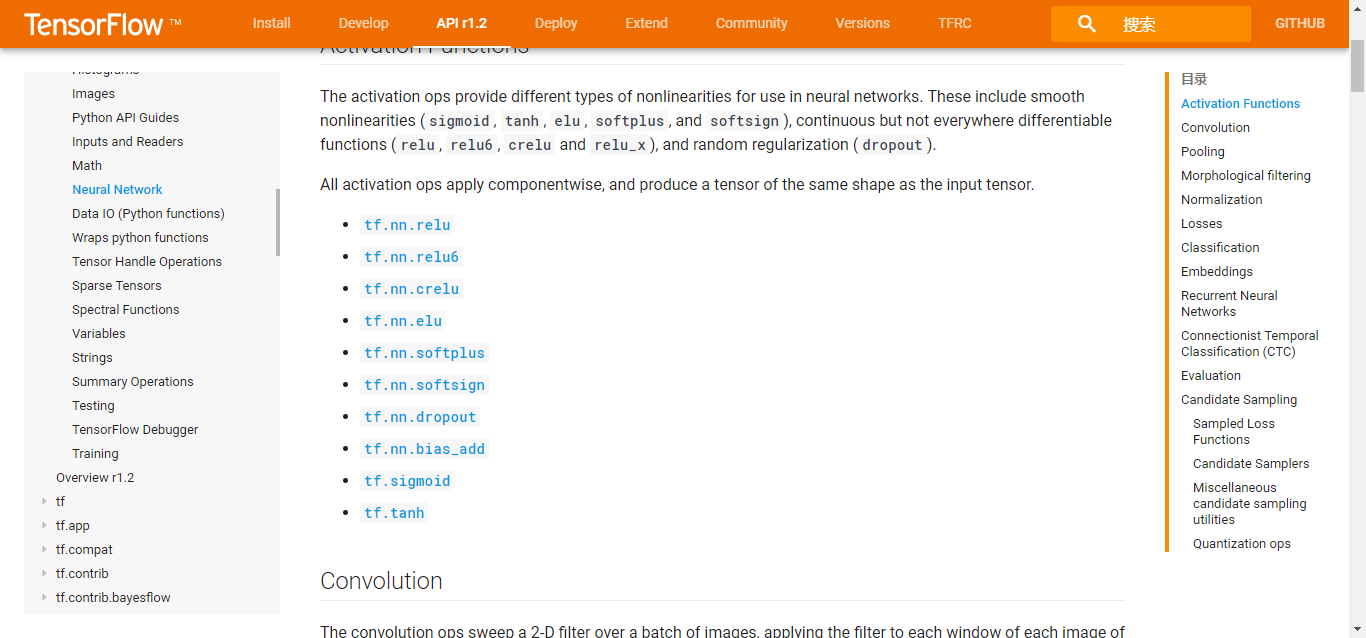
**③Placeholder**

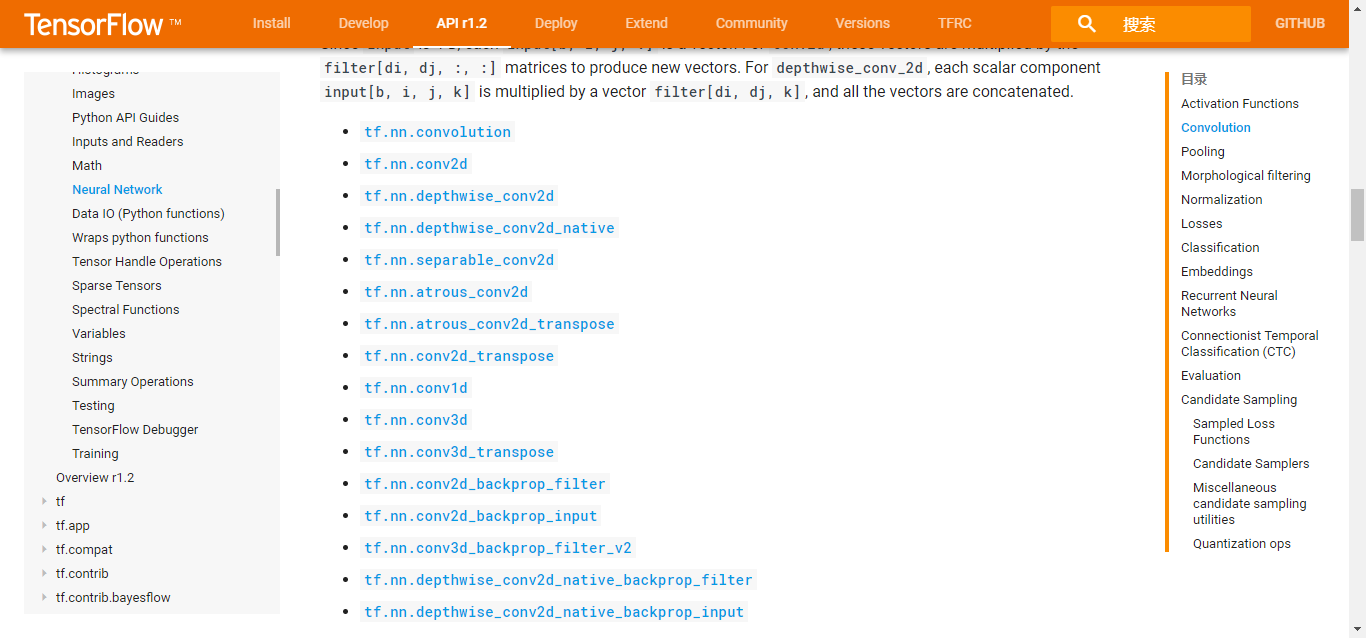
Placeholder在tensorflow中用于保存数据，函数placeholder（type,strucuct…)它的第一个参数是要保存的数据的数据类型，大多数是tensorflow中的**float32**数据类型，后面的参数就是要保存数据的结构。比如要保存一个1×2的矩阵，则struct=[1,2]。它在使用的时候和前面的variable不同的是在session运行阶段，需要给placeholder提供数据，利用feed\_dict的字典结构给placeholder变量“喂数据”。



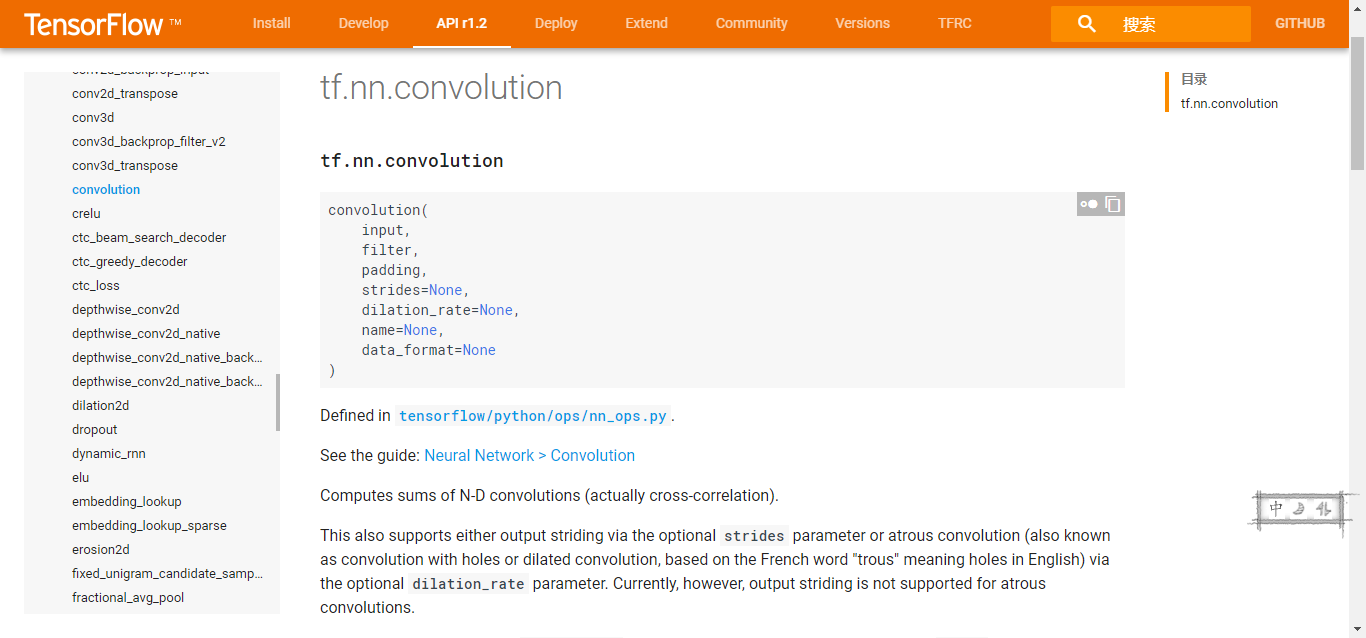
**④Activation functions 了解激励函数**

<https://www.tensorflow.org/api_guides/python/nn>





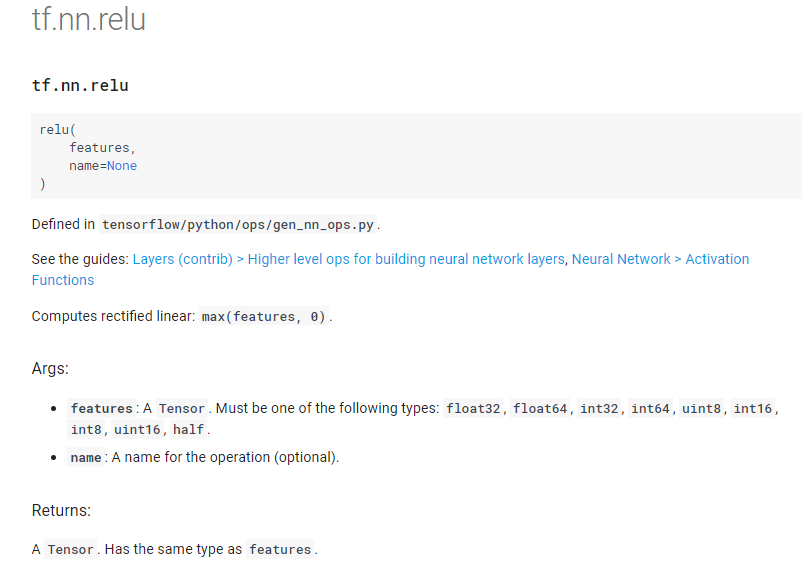
在tensorflow官方网站的文档中可以看到很多种不同的激励函数，它们可以在不同的情况下选择使用，每个激励函数都有对参数的详细解释（此处我没有详细地查看每个函数，打算在以后用到的时候再来找一下）：



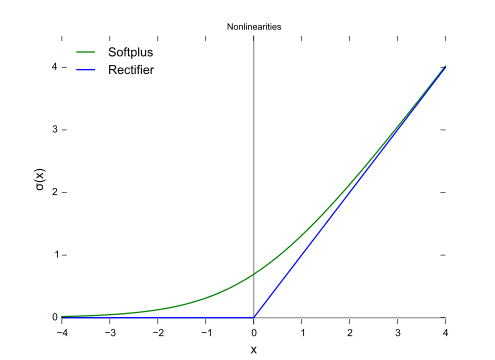
**⑤构建一个完整的简单神经网络**

**（包括添加神经层，计算误差，训练步骤以及判断是否在学习）**

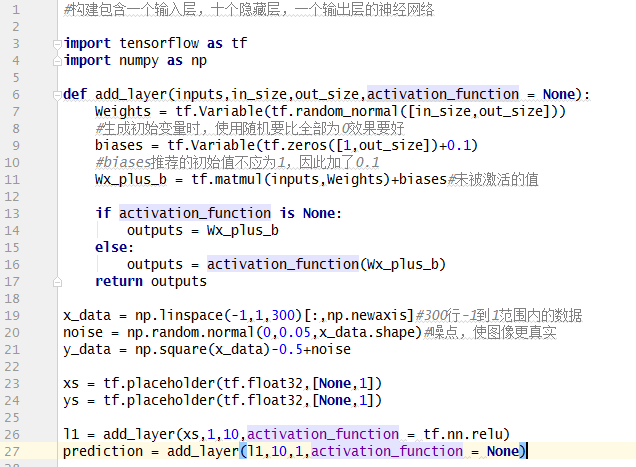
隐藏层选用的激励函数为relu，具体参数如下：

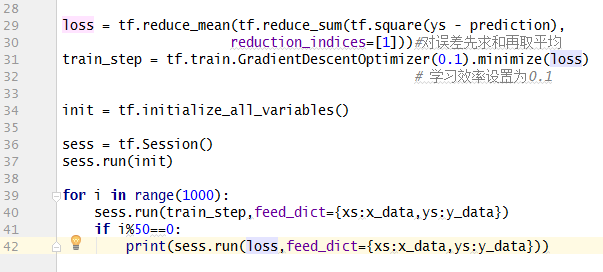


relu函数的图像（蓝色线条）



程序代码：





运行结果显示出了误差在不断缩小：

