一.实验题目

利用CNN模型实现对CIFAR-10数据集的图像分类，利用RNN模型实现对STS Benchmark数据集的语义相似度分析。

二.算法原理

Part1 卷积神经网络

I.lenet5

LeNet5 是一种用于手写体字符识别的非常高效的卷积神经网络，也是十分经典的卷积神经网络架构。由于提出的时间久远，所以相较于这次实验中的其他架构，准确率是最低的。

LeNet5 由两层卷积,池化以及三层全连接层来进行搭建。LeNet5 中，第一个卷积层有 6 个 5*5 的 filter，第二个卷积层有 16 个 5*5 的 filter，每一个卷积层之后加上一层 strides=(2,2)的池化层进行下采样，我选择的是最大池化。同时在卷积层的激活函数上，我采用卷积层通常选择的线性整流函数’relu’，而在最后的全连接层中使用逻辑回归函数’softmax’。

relu 函数:对于进入神经元的来自上一层神经网络的输入向量x，使用线性整流激活函数的神经元会输出

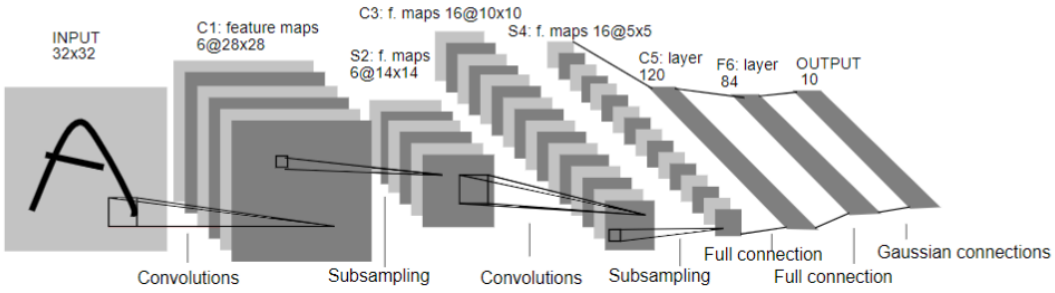
$$\max(0, w^T x + b)$$

至下一层神经元或作为整个神经网络的输出。

Softmax 函数:用来进行离散化概率分布，对于 cifar10 这种分类的问题有很好的效果。

$$S_i = \frac{e_i}{\sum e_j}$$

神经网络结构如下图:



为了提高整个网络在测试集上的准确率我们可以更多的添加 Dropout 层来减少过拟合的程度。在每个训练批次中，通过忽略一半的特征检测器（让一半的隐层节点值为 0），可以明显地减少过拟合现象。这种方式可以减少特征检测器（隐层节点）间的相互作用，检测器相互作用是指某些检测器依赖其他检测器才能发挥作用。通常而言，我们可以在卷积层和池化层之后添加 dropout 层来提高在测试集上的准确率。

2.VGG

VGG 模型相比较于其他的经典架构，最为突出的特点就是其规范和简洁的模块使用方法。对于每一个卷积层而言，选择的卷积核都是 3*3 大小，strides 都为 1，且使用 padding 来使图像的大小不变，且卷积核的数目从 64 个起逐级翻倍增加。而在之后的 maxpooling 层，选择的 poolsize 都为 2*2，且 strides 为 2。

对于激活函数而言卷积层都是使用的’relu’函数，而在最后的全连接层，使用的是’softmax’函数。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

对于 VGG 网络架构而言，可以通过增加深度有效的提高性能，同时在结构上简洁优美，实现起来算不上复杂。但是建立的网络通常规模较为庞大，训练起来较为耗时。

3.Resnet

对于 vgg 网络而言，再在 19 层的深度之上继续加深层数，并不能够有效的对于分类的准确率进行提高，反而招致网络收敛的速度变慢。而 Resnet 网络作者则想到了常规计算机视觉领域常用的 residual representation 的概念，并进一步将它应用在了 CNN 模型的构建当中，于是就有了基本的 residual learning 的 block。它通过使用多个有参层来学习输入输出之间的残差表示，而非像一般 CNN 网络（如 Alexnet/VGG 等）那样使用有参层来直接尝试学习输入、输出之间的映射。

ResNet 准备从这一块动手，假设现在有一个栈的卷积层比如说 2 个卷积层堆叠，将当前这个栈的输入与后面的栈的输入之间的 mapping 称为 underlying mapping，现在的工作是企图替换它引入一种新的 mapping 关系，ResNet 称之为 residual mapping 去替换常规的 mapping 关系。

意思是与其让卷积栈直接拟合 underlying mapping，不如让它去拟合 residual mapping。而 residual mapping 和 underlying mapping 其实是有关联的。

将 underlyingmapping 标记为 H(x)将经过堆叠的非线性层产生的 mapping 标记为 F(x):=H(x)−x。

所以，最原始的 mapping 就被强制转换成 F(x)+x。

然后，作者假设对 residual mapping 的优化要比常规的 underlying mapping 要简单和容易。

而 F(x)+x 在实际的编码过程中，可以被一种叫做快捷连接的结构件来实现。

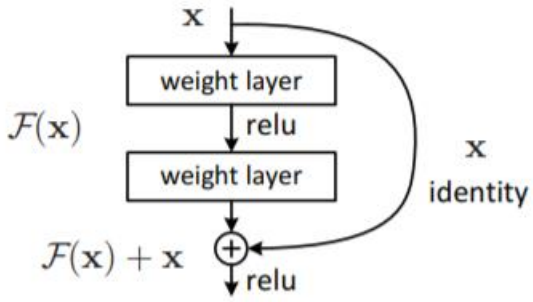
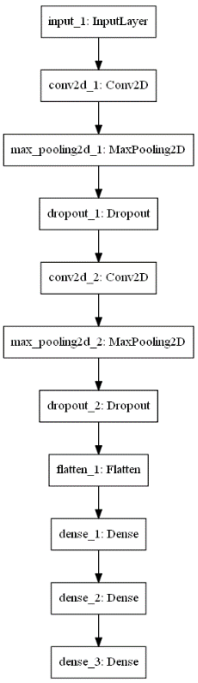


Figure 2. Residual learning: a building block.

快捷连接通常会跳过 1 个或者多个层，在 ResNet 中快捷连接直接运用了 identity mapping，意思就是将一个卷积栈的输入直接与这个卷积栈的输出相加。而 F(x)就是所谓的残差，当 F(x)为 0 时，identity mapping 就是最好的输出结果。而在 keras 的网络搭建过程中，快捷连接编写也十分的简单。

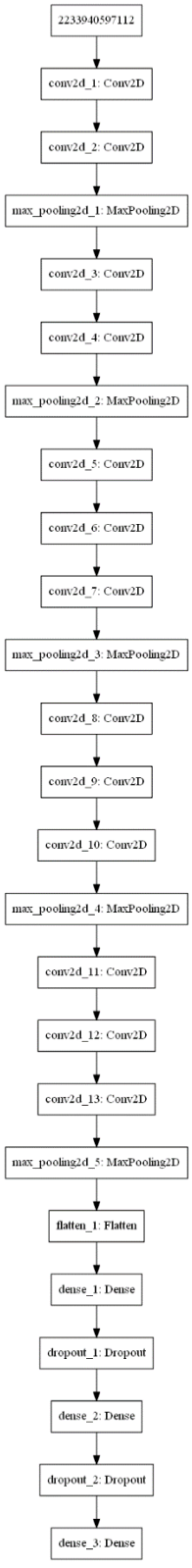
三.网络结构

1.lenet5的网络架构

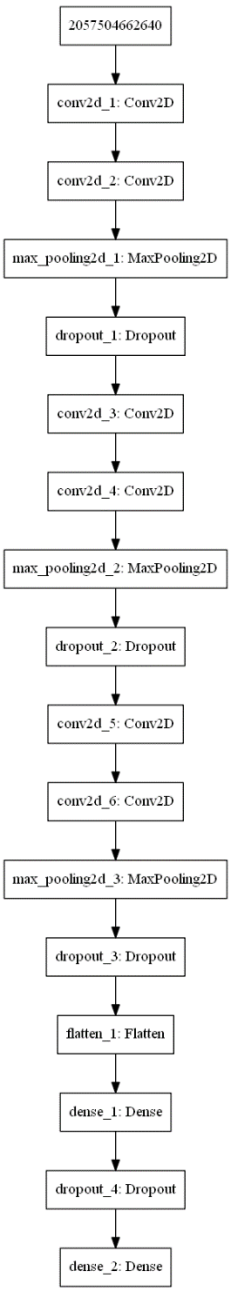


在使用这个lenet5的模型的同时，我在每一个池化层的后面添加了dropout层来一定程度范围内提高在测试集上的准确度，避免由于数据集的过小而导致的过拟合。

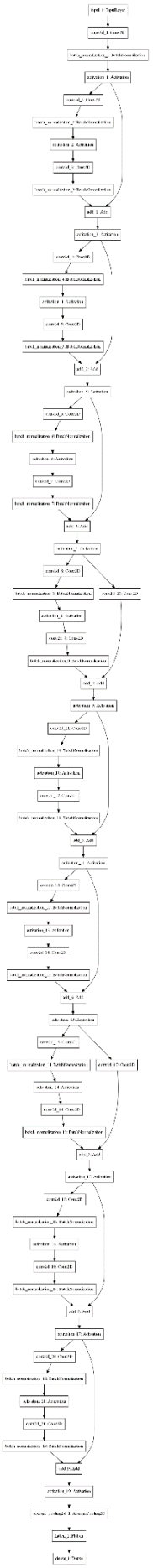
2.vgg16



3.vgg 改动版


2057504662640
conv2d_1: Conv2D
conv2d_2: Conv2D
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_3: Conv2D
conv2d_4: Conv2D
max_pooling2d_2: MaxPooling2D
dropout_2: Dropout
conv2d_5: Conv2D
conv2d_6: Conv2D
max_pooling2d_3: MaxPooling2D
dropout_3: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_4: Dropout
dense_2: Dense

4.Resnet



四.关键代码展示

1.数据处理

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
input_shape = (32,32,3)

x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255


x_train_mean = np.mean(x_train, axis=0)
x_train -= x_train_mean
x_test -= x_train_mean

y_train = keras.utils.to_categorical(y_train,10)
y_test = keras.utils.to_categorical(y_test,10)
```

由于使用的是 keras 架构，我没有进行处理原本数据的，而是直接使用了 keras 中 dataset 部分的 loaddata()模块将数据直接导出。在训练集中，总共提供了 50000 张图，每张图的 shape 为(32,32,3)，对于每一张图有一个 label。而在测试集中，总共提供了 10000 张图片。

同时对于图像进行归一化(/255)，之后进行标准化以期提高准确率。再之后将 label 编程 one_hot 矩阵，以供后来 Softmax 分类。

2.lenet5 网络建立

```
def lenet5(input_shape):
    inputs = Input(shape=input_shape)
    x = Conv2D(6,kernel_size=5,strides=1,activation='relu',ker
nel_initializer='he_normal')(inputs)
    x = MaxPooling2D((2,2),strides=(2,2))(x)
    x = Dropout(0.2)(x)
    x = Conv2D(16,kernel_size=5,strides=1,activation='relu',ke
rnel_initializer='he_normal')(x)
    x = MaxPooling2D((2,2),strides=(2,2))(x)
    x = Dropout(0.2)(x)
    x = Flatten()(x)
    x = Dense(120,activation='relu',kernel_initializer='he_nor
mal')(x)
    x = Dense(84,activation='relu',kernel_initializer='he_norm
al')(x)
    outputs = Dense(10,activation='softmax',kernel_initializer
='he_normal')(x)
```

```
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

LeNet5 中，第一个卷积层有 6 个 5*5 的 filter，第二个卷积层有 16 个 5*5 的 filter，每一个卷积层之后加上一层 strides=(2,2)的池化层进行下采样。为了一定程度上减轻过拟合的影响，在池化层之后，我都添加了 Dropout 层。之后按照论文中的模型，依次添加三层全连接层。

3.vgg16

```
model = Sequential()
#model.add(Input(shape=(32,32,3)))
model.add(Conv2D(64, (3, 3), strides=(1, 1), input_shape=(32,
32, 3), padding='same', activation='relu',
                 kernel_initializer='uniform'))
model.add(Conv2D(64, (3, 3), strides=(1, 1), padding='same', a
ctivation='relu', kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 2), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(128, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(256, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(256, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same',
activation='relu', kernel_initializer='uniform'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

根据论文中的模型进行搭建，卷积核都为 3*3 大小，且 strides 为 1。最大池化层选择的 poolsize=(2,2)。对于卷积层的 filter 数，首层选择 64 个，逐块翻倍递增。且在每一个卷积层将 padding 设为 same。

3.vgg 改动版

```
model=Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=(32,32
,3),kernel_initializer='he_uniform', padding='same'))
model.add(Conv2D(64,(3,3),activation='relu',kernel_initializer
='he_uniform', padding='same'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(128,(3,3),activation='relu',kernel_initialize
r='he_uniform', padding='same'))
model.add(Conv2D(128,(3,3),activation='relu',kernel_initialize
r='he_uniform', padding='same'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(256, (3, 3), activation='relu', kernel_initia
lizer='he_uniform', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', kernel_initia
lizer='he_uniform', padding='same'))
model.add(MaxPooling2D((2,2)))

model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))
```

由于 vgg16 网络模型较为庞大，且其收敛速度较为缓慢，可以采用较为缩小的模型来加快速度进行训练。在前面三个模块的的选择和 vgg 经典模型类似，都是 filter 数分别为 64，128，256，且池化层的 poolsize 选择相同，同时还在池化层之后添加了 Dropout 层。但是总共只有 6 个卷积层(2 个 64，2 个 128，2 个 256)。之后有两个全连接层。

4.Resnet

```
def block(inputs,num_filters=16,kernel_size=3,strides=1,activa
tion='relu'):
    x = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))(inputs)
    x = BatchNormalization()(x)
    if activation != None:
        x = Activation('relu')(x)
    return x

def resnet(input_shape):
    inputs = Input(shape=input_shape)# Input 层，用来当做占位使用
    x = block(inputs)
    for i in range(6):
        a = block(inputs = x)
        b = block(inputs=a,activation=None)
        x = keras.layers.add([x,b])
        x = Activation('relu')(x)
    for i in range(6):
        strides = 1
        if i == 0:
            strides = 2
        a = block(inputs=x,num_filters=32,strides=strides)
        b = block(inputs=a,activation=None,num_filters=32)
        if i == 0:
            x = Conv2D(32,
                  kernel_size=3,
                  strides=2,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))(x)
        x = keras.layers.add([x,b])
        x = Activation('relu')(x)
    for i in range(6):
        strides = 1
        if i == 0:
```

```
            strides = 2
        a = block(inputs=x,num_filters=64,strides=strides)
        b = block(inputs=a,activation=None,num_filters=64)
        if i == 0:
            x = Conv2D(64,
                  kernel_size=3,
                  strides=2,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))(x)
        x = keras.layers.add([x,b])
        x = Activation('relu')(x)
    x = AveragePooling2D(pool_size=2)(x)
    # out:4*4*64
    y = Flatten()(x)
    # out:1024
    outputs = Dense(10,activation='softmax',
                    kernel_initializer='he_normal')(y)
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

首先需要建立的是一个 block 模块，其中包括一个卷积层和一个 BN 层，是否需要激活进行判断。

而在整体上从上到下依次为输入层，6 层 filter 数为 16 的 block，6 层 filter 数为 32 的 block，6 层 filter 数为 64 的 block，全连接层，输出层。

在第一个 6 层中，strides=1，而在之后的两个 6 层中，首层 strides=2。同时为了快捷链接，在后两个 6 层中需要对输入进行一次卷积操作之后，使其 shape 和正常的卷积层的输出一样时，才能够进行 add 操作。

5.callbacks

```
checkpoint = ModelCheckpoint(filepath=filepath,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True)

lr_scheduler = LearningRateScheduler(lr_schedule)

lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               min_lr=0.5e-6)

callbacks = [checkpoint, lr_reducer, lr_scheduler]
```

其中 modelcheckpoint 的内容如下。

ModelCheckpoint [source]

```
keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_best_only=False, save_weig
```

在每个训练期之后保存模型。

`filepath` 可以包括命名格式选项，可以由 `epoch` 的值和 `logs` 的键（由 `on_epoch_end` 参数传递）来填充。

例如：如果 `filepath` 是 `weights.{epoch:02d}-{val_loss:.2f}.hdf5`， 那么模型被保存的的文件名就会有训练轮数和验证损失。

参数

- **filepath**: 字符串，保存模型的路径。
- **monitor**: 被监测的数据。
- **verbose**: 详细信息模式，0 或者 1 。
- **save_best_only**: 如果 `save_best_only=True`， 被监测数据的最佳模型就不会被覆盖。
- **mode**: {auto, min, max} 的其中之一。 如果 `save_best_only=True`，那么是否覆盖保存文件的决定就取决于被监测数据的最大或者最小值。 对于 `val_acc`，模式就会是 `max`，而对于 `val_loss`，模式就需要是 `min`，等等。 在 `auto` 模式中，方向会自动从被监测的数据的名字中判断出来。
- **save_weights_only**: 如果 True，那么只有模型的权重会被保存 (`model.save_weights(filepath)`)，否则的话，整个模型会被保存 (`model.save(filepath)`)。

Learningratescheduler 如下：

LearningRateScheduler [source]

```
keras.callbacks.LearningRateScheduler(schedule, verbose=0)
```

学习速率定时器。

参数

- **schedule**: 一个函数，接受轮索引数作为输入（整数，从 0 开始迭代） 然后返回一个学习速率作为输出（浮点数）。
- **verbose**: 整数。 0：安静，1：更新信息。

ReduceLROnPlateau 如下：

ReduceLROnPlateau [source]

```
keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10, verbose=0, mode='auto', r
```

当标准评估停止提升时，降低学习速率。

当学习停止时，模型总是会受益于降低 2-10 倍的学习速率。 这个回调函数监测一个数据并且当这个数据在一定「有耐心」的训练轮之后还没有进步， 那么学习速率就会被降低。

例子

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=5, min_lr=0.001)
model.fit(X_train, Y_train, callbacks=[reduce_lr])
```

参数

- **monitor**: 被监测的数据。
- **factor**: 学习速率被降低的因数。新的学习速率 = 学习速率 * 因数
- **patience**: 没有进步的训练轮数，在这之后训练速率会被降低。
- **verbose**: 整数。0：安静，1：更新信息。
- **mode**: {auto, min, max} 其中之一。如果是 `min` 模式，学习速率会被降低如果被监测的数据已经停止下降； 在 `max` 模式，学习塑料会被降低如果被监测的数据已经停止上升； 在 `auto` 模式，方向会被从被监测的数据中自动推断出来。
- **min_delta**: 对于测量新的最优化的阀值，只关注巨大的改变。
- **cooldown**: 在学习速率被降低之后，重新恢复正常操作之前等待的训练轮数量。
- **min_lr**: 学习速率的下边界。

6.数据增强部分

```
datagen = ImageDataGenerator(
    # set input mean to 0 over the dataset
    featurewise_center=False,
    # set each sample mean to 0
    samplewise_center=False,
    # divide inputs by std of dataset
    featurewise_std_normalization=False,
    # divide each input by its std
    samplewise_std_normalization=False,
    # apply ZCA whitening
    zca_whitening=False,
    # epsilon for ZCA whitening
    zca_epsilon=1e-06,
    # randomly rotate images in the range (deg 0 to 180)
    rotation_range=0,
    # randomly shift images horizontally
    width_shift_range=0.1,
    # randomly shift images vertically
    height_shift_range=0.1,
    # set range for random shear
    shear_range=0.,
    # set range for random zoom
    zoom_range=0.,
    # set range for random channel shifts
```

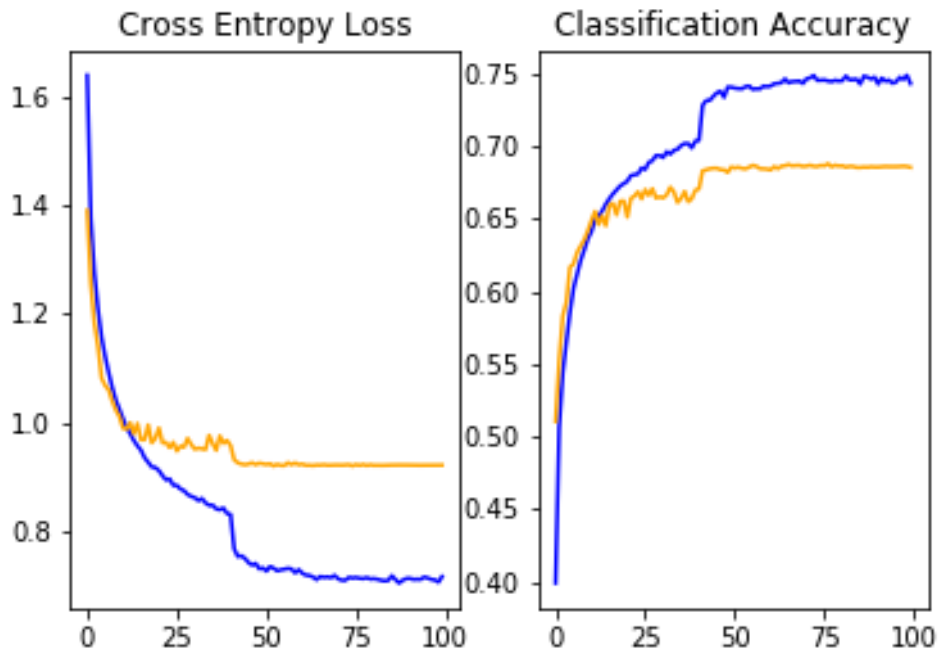
```
    channel_shift_range=0.,
    # set mode for filling points outside the input boundaries
    fill_mode='nearest',
    # value used for fill_mode = "constant"
    cval=0.,
    # randomly flip images
    horizontal_flip=True,
    # randomly flip images
    vertical_flip=False,
    # set rescaling factor (applied before any other transform
ation)
    rescale=None,
    # set function that will be applied on each input
    preprocessing_function=None,
    # image data format, either "channels_first" or "channels_
last"
    data_format=None,
    # fraction of images reserved for validation (strictly bet
ween 0 and 1)
    validation_split=0.0)

# Compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is app
lied).
datagen.fit(x_train)

# Fit the model on the batches generated by datagen.flow().
model.fit_generator(datagen.flow(x_train, y_train, batch_size=
64),
                    validation_data=(x_test, y_test),
                    epochs=100, verbose=1, workers=4,
                    callbacks=callbacks,steps_per_epoch=x_trai
n.shape[0])
```

由于cifar10的训练集过小，众多神经网络在训练的时候都会存在过拟合的情况，所以通过图像增广技术，我们可以更多的来扩充数据集，极大程度的减少过拟合的程度，也极大的提高了在测试上的准确率。上面的这段代码应用自keras的中文官方文档，我只是照搬下来运用。

五.结果分析展示

1.lenet5

Lenet5作为提出时间较早的模型，结构层次比较的简单，所以相对而言可能准确率比不上后面提出的这些模型。



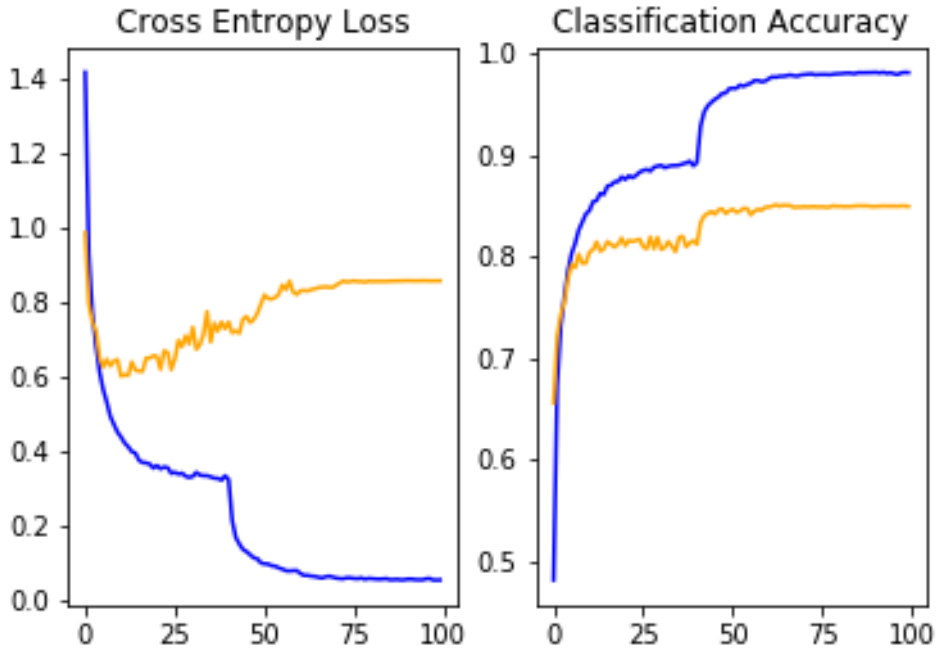
上图是lenet5模型在epoch=100时的训练过程中的loss(blue), val_loss(orange), accuracy(blue),val_accuracy(orange)的变化曲线，其中加了val前缀的是在测试集中的表现。就准确率而言，最后的准确率也没有超过0.7，并且可以明显的看出过拟合情况，在epoch>50之后，准确率几乎没有大的增加。

准确率如下：

```
10000/10000 [==============================] - 1s 91us/step
Test loss: 0.9214602243423462
Test accuracy: 0.6855000257492065
```

2.vgg改动版

在进行了vgg16的实验时，我发现由于网络规模较大单个epoch耗时较长，且在进行训练时收敛的速度较慢，所以我在实际进行测试时改变了原本的vgg模型(代码见上文)，收敛速度较快，但是可能最后的准确率不能够算太高。



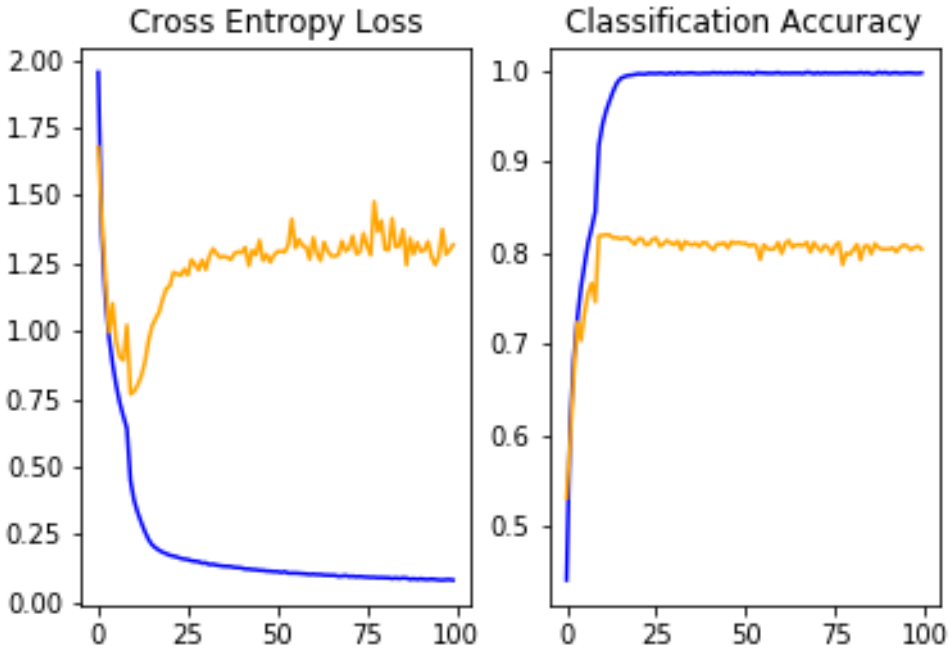
在100个epoch下，可以发现在50epoch左右时，在训练集上的准确率得到了一个大的提升，同时loss也大幅减小，但是对于在整个测试集上的准确率没有了太大的提升。也可以看出更为严重的过拟合问题。

准确率如下：

```
10000/10000 [==============================] - 4s 449us/step
Test loss: 0.8320876037359237
Test accuracy: 0.8513000011444092
```

3.resnet不加数据加强

我所运行的resnet网络就是最为简单的resnet18网络，第一次运行这个大名鼎鼎的网络时我无奈的发现没有加上数据增强的resnet的测试集上的准确率还远远比不上进行改动了的vgg网络。



在这张运行的结果展示图上，我们可以十分惊喜的发现在训练集上的准确率已经几乎达到了1.00的程度，而且是在不到25个epoch时就达到这样的效果。但是我们不是需要训练集上的效果，过拟合的程度实在是太高了。最后的准确率还比不上改动的vgg模型。

准确率如下:

```
10000/10000 [==============================] - 5s 517us/step
Test loss: 1.3180758610725403
Test accuracy: 0.8043000102043152
```

4.resnet添加数据加强

由于实在不甘心resnet运行的效果如此的差，过拟合程度如此的高，最后我进行了数据加强的处理。通过更多的训练集来得到过拟合程度的减少。由于我在实验的过程中都是没有保存训练中的数据的，(上面模型的history都是我重新测试才得到的)，又由于添加了数据加强后的resnet运行时间太过漫长，所以这里我只有最后保存好的权重文件。
在训练的过程中，我改用epoch=10来进行训练(GTX 1050Ti)，根据最后的测试集上的准确率可以看出过拟合的程度大大减小。

准确率如下:

```
10000/10000 [==============================] - 4s 437us/step
Test loss: 0.45399030742645263
Test accuracy: 0.9153000116348267
```

六.Reference

[1]https://arxiv.org/pdf/1512.03385.pdf

[2] Yann LeCun,Leon Bottou,Yoshua Bengio,Patrick Haner “GradientBased Learning Applied to Document Recognition ”

[3] https://arxiv.org/pdf/1409.1556.pdf

[4]https://blog.csdn.net/briblue/article/details/83544381

[5] https://keras.io/zh/

[6] https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

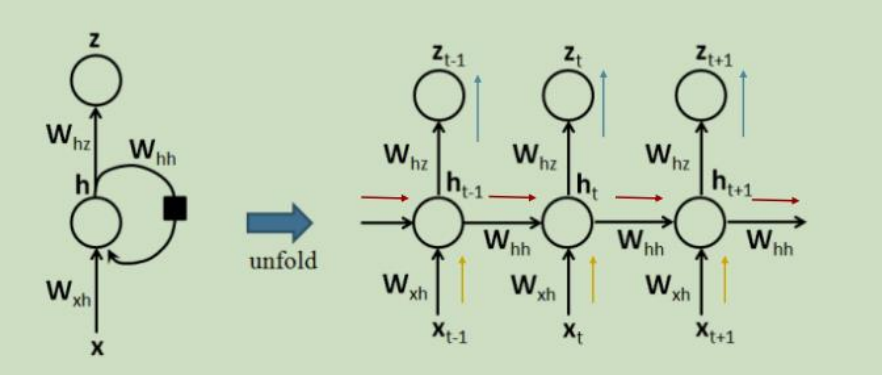
一. 实验题目

利用 RNN 模型实现对 STS Benchmark 数据集的语义相似度分析。

二. 实验原理

1. RNN 基础介绍

RNN 主要针对的是序列数据，通过神经网络在时序上的展开，找到样本之间的序列相关性。基本的 RNN 网络如图：



对于 RNN 中的其中一个神经元，它随着时序的展开，会结合之前的学习到的，以及新的输入，来更新自己的权重。神经元在时序上的迭代公式为(注意，在时序上共享权重)：

$$\boldsymbol{h}_t = f(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h),\ f\ is\ one\ kind\ of\ activation\ functions.$$

$$\boldsymbol{z}_t = softmax(\boldsymbol{W}_{hz}\boldsymbol{h}_t + \boldsymbol{b}_z)$$

然后通过定义一个总损失函数，对权重进行求导，并且沿着梯度的反方向更新，这就是反向传播的过程，由于 RNN 是一个时序演化过程，所以对有些权重的求导会用到之前的输出，所以 RNN 的反向传播叫做 BPTT。

接下来我们写下求导的过程，还是利用上图，设在 t时刻的损失函数与$\boldsymbol{z}_t$（预测值）和$\boldsymbol{y}_t$（真实值）有关，即$loss = loss(\boldsymbol{z}_t, \boldsymbol{y}_t)$。那么对$\boldsymbol{W}_{hz}$的求导为：

$$\frac{\partial loss}{\partial \boldsymbol{W}_{hz}} = \frac{\partial loss}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{W}_{hz}}$$

对$\boldsymbol{W}_{hh}$的求导为：

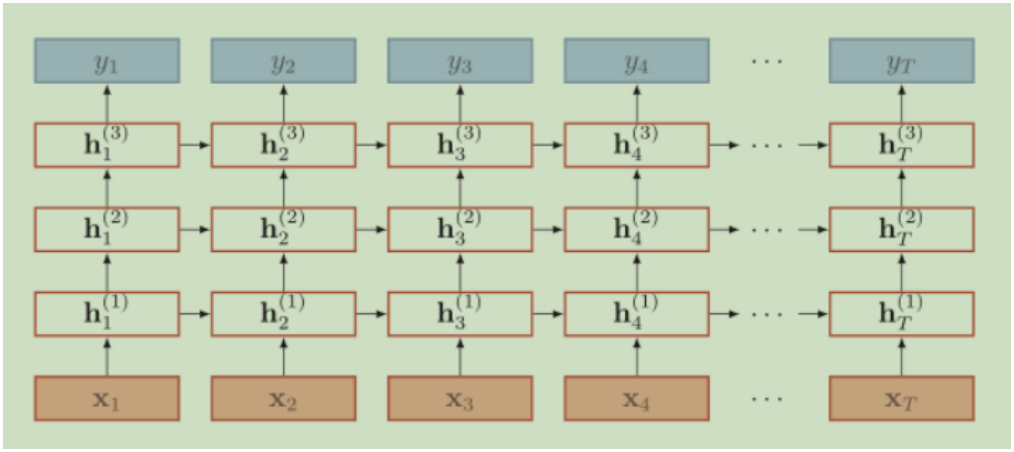
$$\frac{\partial loss}{\partial \boldsymbol{W}_{hh}} = \sum_{i=0}^{t}\frac{\partial loss}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_t}\left(\prod_{j=i+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}\right)\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{W}_{hh}}$$

对$\boldsymbol{W}_{xh}$的求导为：

$$\frac{\partial loss}{\partial \boldsymbol{W}_{xh}} = \sum_{i=0}^{t}\frac{\partial loss}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_t}\left(\prod_{j=i+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}\right)\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{W}_{xh}}$$

2. 加强版 RNN：

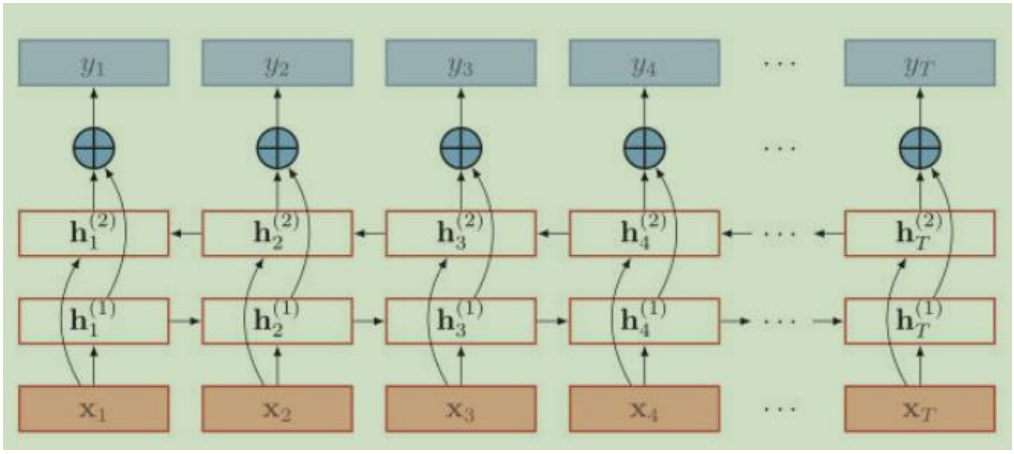
A. 像全连接层一样，RNN 也可以多层堆叠，堆叠的 RNN 叫做(Stacked RNN). 模型如下：



优点：加强网络的可表示性，即可以表示更多非线性关系；

缺点：层数过多则参数更难优化，并且容易出现梯度消失或梯度爆炸。

B. 双向 RNN，既然可以正方向训练，也可以反着训练：



优点：相比于单向的 RNN，双向的 RNN 显然可以发现更多的相关性。

3. 解决梯度消失或梯度爆炸的问题：

在(1)RNN 基础介绍中，我们写出了损失函数$loss$对各个权重的求导公式，这里再搬过来：

$$\frac{\partial loss}{\partial \boldsymbol{W}_{hz}} = \frac{\partial loss}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{W}_{hz}}$$

$$\frac{\partial loss}{\partial \boldsymbol{W}_{hh}} = \sum_{i=0}^{t}\frac{\partial loss}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_t}\left(\prod_{j=i+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}\right)\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{W}_{hh}}$$

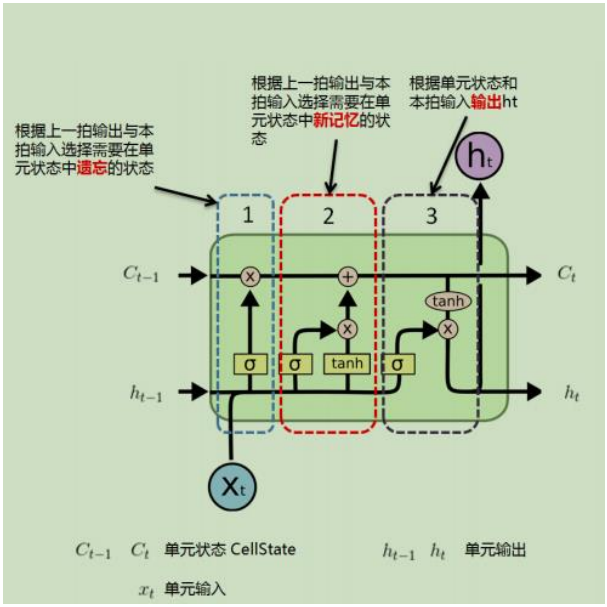
$$\frac{\partial loss}{\partial \boldsymbol{W}_{xh}} = \sum_{i=0}^{t}\frac{\partial loss}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_t}\left(\prod_{j=i+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}\right)\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{W}_{xh}}$$

$\prod_{j=k+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{t-1}} = \tanh' \boldsymbol{W}$这个地方就很容易出现问题，如果$\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}$过大，那$\prod_{j=k+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{t-1}}$就会很

大导致梯度爆炸，同理，如果$\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}$过小，那$\prod_{j=k+1}^{t}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{t-1}}$就会很小，导致梯度消失。所以为了解决这个问题，就提出了两种改进的结构，不再是上面那张图的单个神经元那

种结构，而是称作LSTM和GRU的结构，而GRU是LSTM的改进版，所以这里只介绍LSTM。

LSTM的结构如图：



图中1、2、3分别对应LSTM结构的遗忘门、输入门和输出门。

遗忘门根据上一个C_{i-1}选择当前单元状态的输入的每部分需要保留的程度，公式为：

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right)$$

$$C_{t1} = f_t * C_{t-1}$$

输入门先将输入通过两个激活函数，然后点乘获得当前产生的隐状态有多少需要保留并加入到下一个C_i，公式为：

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(W_C \cdot \left[h_{t-1}, x_t\right] + b_C\right)$$

$$C_{t2} = i_t * \tilde{C}_t$$

此时可以更新细胞状态(Cell State)：$C_t = C_{t1} + C_{t2}$

输出门将输出通过激活函数，再与新的细胞状态点乘得到当前单元状态的输出：

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

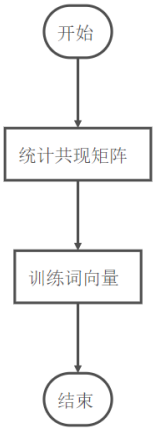
此时我们再来看下$\prod_{j=k+1}^{t} \frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{t-1}}$，由于这里的$\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{t-1}} = \tanh' \sigma$, 而这值不是0就是1，那么$\prod_{j=k+1}^{t} \frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{t-1}}$也是不是0就是1，所以就解决了梯度消失或者梯度爆炸的问题。

4. 将句子转化为向量表示：

以上是我结合课件和网上之后的总结，搞懂了网络的原理之后，就是如何将两个句子转化为向量表示， 有两个选择，一个是使用Glove训练好的词向量，另一个是使用Word2Vec自己训练自己的词向量模型。

A. Glove 原理：

首先，看下 Glove 模型：



设共现矩阵为X，则X_{ij}表示为单词 i 和单词 j 共同出现在一个窗口内的次数。作者定义的代价函数如下：

$$J = \sum_{i,j}^{N} f(X_{ij})(v_i^T v_j + b_i + b_j - \log(X_{ij}))^2$$

v_i和v_j是单词i和单词j的词向量，b_i和b_j是两个偏差项，f是权重函数，N是词汇表的大小。作者认为一个值$\frac{X_{ik}}{X_{jk}}$，该值能由共现矩阵获得，如果能让词向量通过某种计算获得，就说明词向量蕴含了共现矩阵的信息，所以作者定义的代价函数的原型是：

$$J = \sum_{i,j,k}^{N} (\frac{X_{ik}}{X_{jk}} - g(v_i, v_j, v_k))^2$$

但由于复杂度太高，所以通过一系列推导，就得到上面的代价函数，这里就不再赘述。与Word2Vec不同，Glove没有用到神经网络，每次训练通过adgrad优化方法去更新词向量以及其他参数。

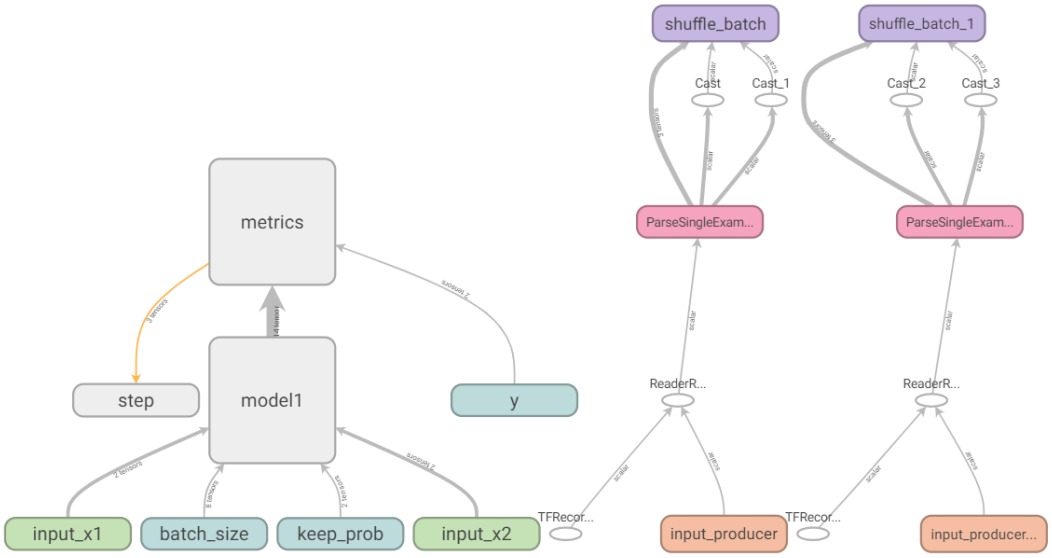
B. Word2Vec 原理：

Word2Vec 是利用神经网络来训练获得词向量的，它是取中间隐层的参数矩阵来表示词向量，比如你想要用 300 维向量来表示 10000 个词语中的每个词语，那么中间隐层的参数矩阵大小就为[10000, 300]，这个矩阵就是表示这 10000 个词语的所有词向量。通过神经网络训练，使得这个矩阵的参数满足一定要求，则可以表达词语。该网络的输入层是词语的 one_hot 向量表示，为 10000 维向量，输出层也是 10000 维向量，每一维的数字对应该维对应的单词与输入词同时出现在上下文的概率。Word2Vec 有两种模型，一种是 Skip-Gram，另一种是 CBOW，Skip-Gram 是利用中心词来预测上下文，而 CBOW 是利用上下文来预测中心词，具体例子将在 PPT 上展示，这里就不再放图放例子了。

两个模型相比，skip-gram 模型能产生更多训练样本，抓住更多词与词之间语义上的细节，在语料足够多足够好的理想条件下，skip-gram 模型是优于 CBOW 模型的。在语料较少的情况下，难以抓住足够多词与词之间的细节，CBOW 模型求平均的特性，反而效果可能更好。

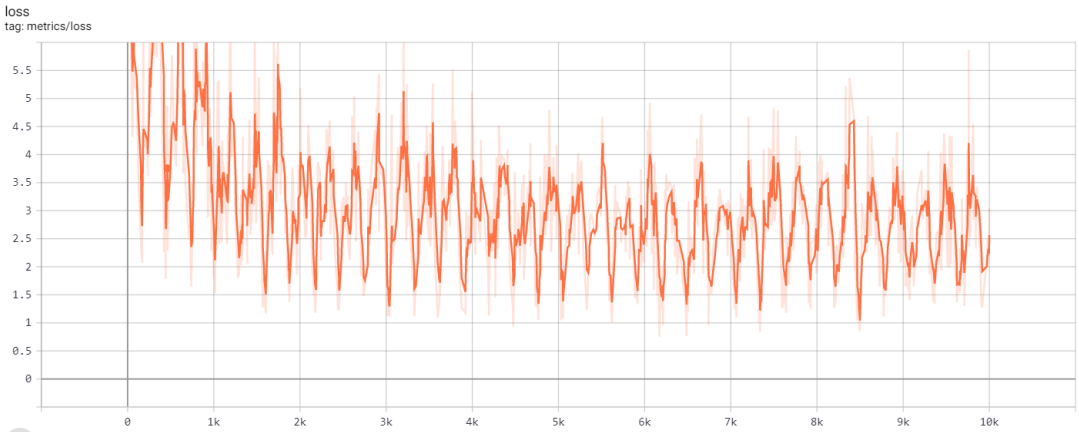
三. 网络结构

1. 只有 LSTM 单种结构，利用 tensorboard 画出基本的框架图。如下：

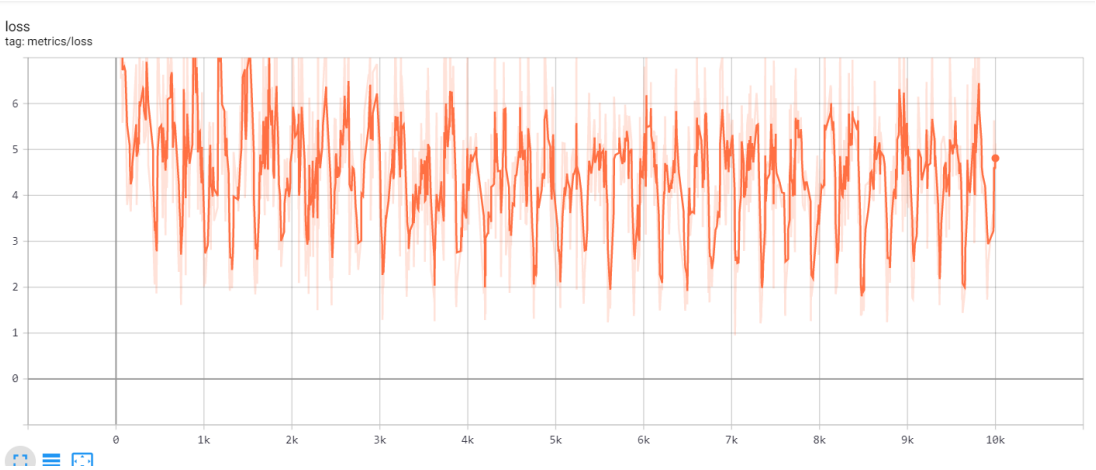


两个句向量通过孪生网络，得到两个向量，然后通过余弦相似度求出相似度，只有与真实的相似度作差求平方，得到 loss，最后将 loss 除以样本数量，这个 loss 就是均方误差 mse。

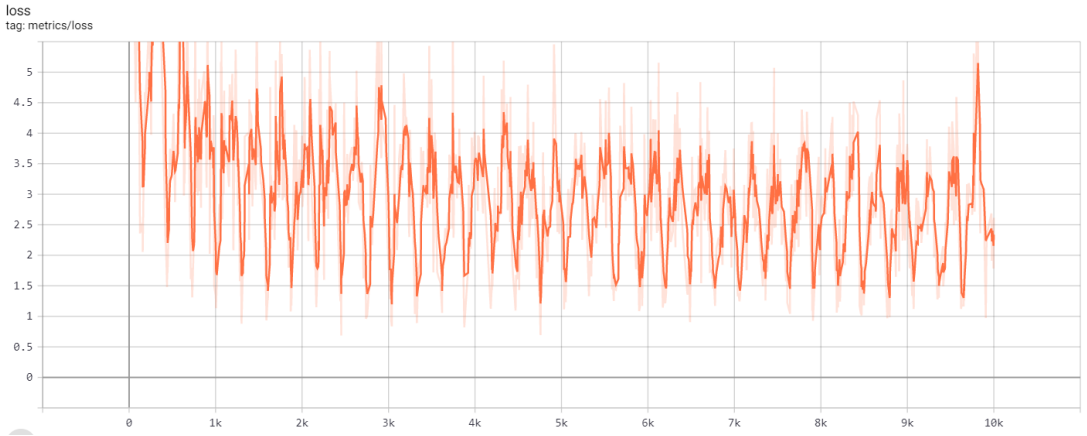
1） 单层单向 LSTM 的 train_loos 随着训练次数的图像：



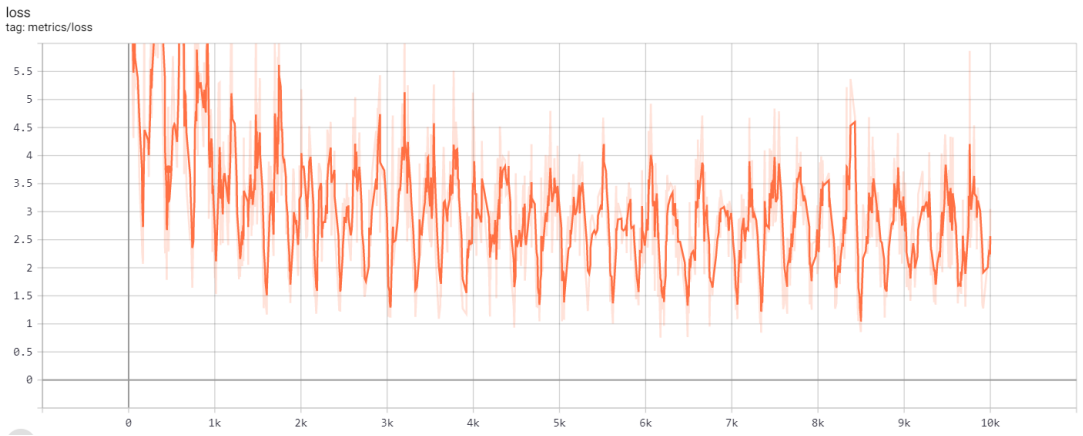
2） 多层单向 LSTM 的 train_loos 随着训练次数的图像：



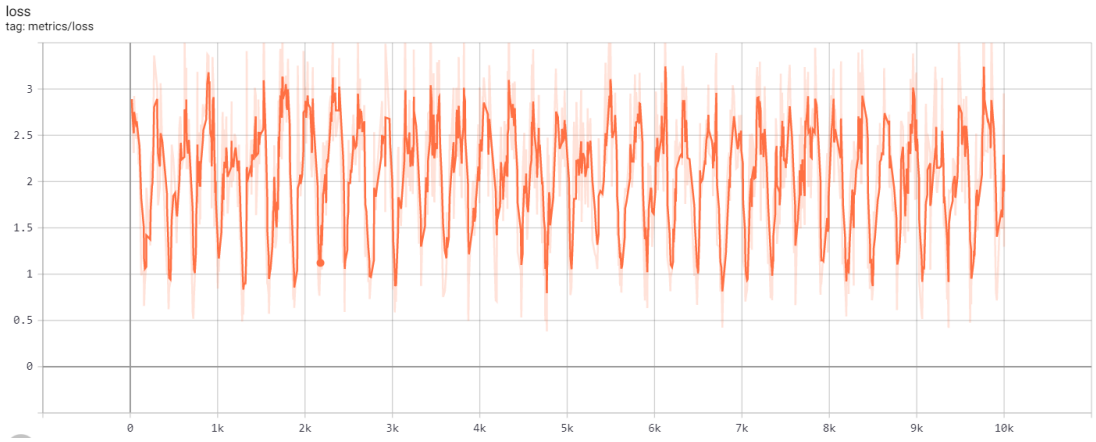
3） 单层双向 LSTM 的 train_loos 随着训练次数的图像：



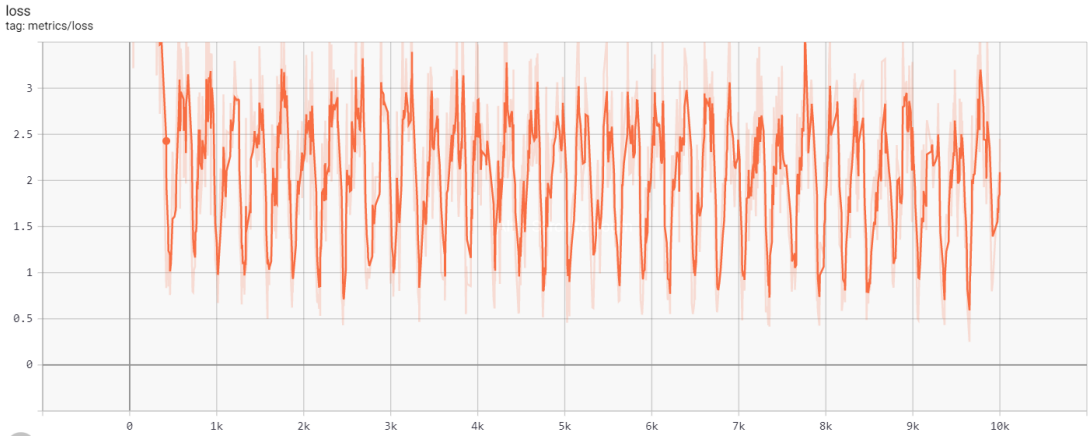
4） 多层双向 LSTM 的 train_loos 随着训练次数的图像：



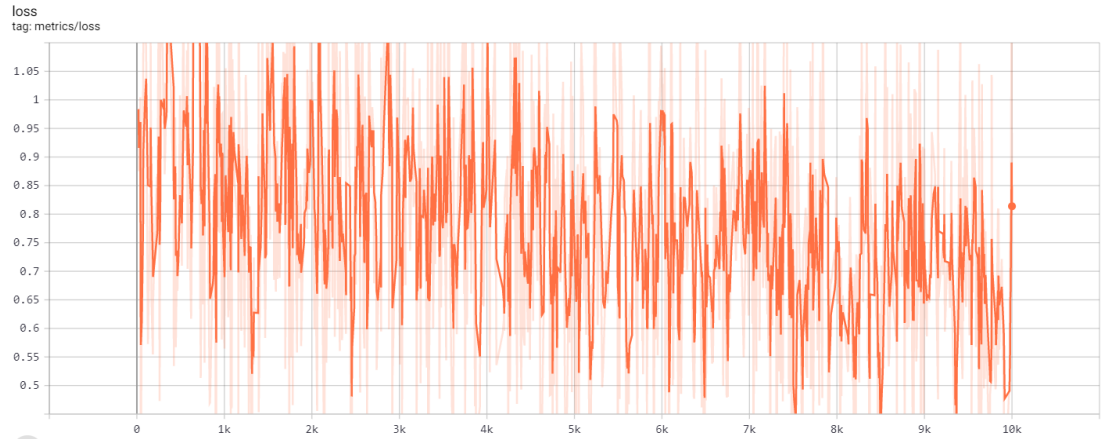
2. 在 LSTM 后面加一层全连接层，只展示在单层单向 LSTM 添加后的 loss 图像：



3. 两个句向量通过孪生神经网络（RNN+一层全连接）后，拼接在一起，经过两层全连接，获得相似度，损失函数是均方误差，loss 图像为：



4. 在上面模型的基础上，损失函数改为 1-皮尔森相关系数（一定样本的预测相似度与这些样本的真实相似度的皮尔森相关系数），loss 图像为：



以上网络模型是我尝试的一部分网络模型，其他网络模型效果也都差不多，但是训练好的网络最后在测试集上的均方误差都是在1.5到2.3，效果都不是太好。并且在训练集上的loss震荡得很厉害（可能是我某个地方写错了，但我找了好多天都没找到），但是在测试集上的loss却是会收敛的。至于在测试集上预测的相似性和真实的相似性的皮尔森相关系数我就没有统计了，因为个人觉得皮尔森相关系数越大，预测的也不一定准，因为相关系数只是向量之间的关系，即使相关系数大，可能预测的相似性与真实的相似性差距还是很大的。

四． 创新

1. 数据集的制作：

神经网络的喂入总是批量喂入的，所以得解决如何批量获取句向量（由词向量组成）以及它们相似度的问题。我利用的是制作 tfrecord 以及使用 shuffle_batch 函数实现批量获取。

制作数据集的部分代码如下（不完整）：

```
        example = tf.compat.v1.train.Example(features=tf.compat.v1.train.Features(feature={
            'similarity': tf.compat.v1.train.Feature(
                float_list=tf.compat.v1.train.FloatList(value=[similarity])),
            'length_0': tf.compat.v1.train.Feature(
                int64_list=tf.compat.v1.train.Int64List(value=[length[0]])),
            'sentence_vector_0': tf.compat.v1.train.Feature(
                float_list=tf.compat.v1.train.FloatList(value=sentence_vector[0][:Max_Word_Num * Dimension])),
            'length_1': tf.compat.v1.train.Feature(
                int64_list=tf.compat.v1.train.Int64List(value=[length[1]])),
            'sentence_vector_1': tf.compat.v1.train.Feature(
                float_list=tf.compat.v1.train.FloatList(value=sentence_vector[1][:Max_Word_Num * Dimension]))
            }))
        writer.write(example.SerializeToString())
```

该部分是在一个 for 循环中，每次写入两个句子的相似度、两个句子的实际长度（句子长短不一，有些会补 0）以及两个句子对应的句向量。

批量获取数据集的代码如下（完整）：

```
def read_TFRecord(TFRcord_path):
    reader = tf.TFRecordReader()
    filename_queue = tf.compat.v1.train.string_input_producer([TFRcord_path], shuffle=True)

    # 解析一个example
    _, serialize_example = reader.read(filename_queue)
    features = tf.io.parse_single_example(serialize_example, features={
        'similarity': tf.io.FixedLenFeature([1], tf.float32),
        'length_0': tf.io.FixedLenFeature([], tf.int64),
        'sentence_vector_0': tf.io.FixedLenFeature([Max_Word_Num * Dimension], tf.float32),
        'length_1': tf.io.FixedLenFeature([], tf.int64),
        'sentence_vector_1': tf.io.FixedLenFeature([Max_Word_Num * Dimension], tf.float32)
        })
    similarity = features['similarity']
    length_0 = tf.cast(features['length_0'], tf.int32)
    sentence_vector_0 = features['sentence_vector_0']
    length_1 = tf.cast(features['length_1'], tf.int32)
    sentence_vector_1 = features['sentence_vector_1']
    return similarity, length_0, sentence_vector_0, length_1, sentence_vector_1
```

```
def get(num, istrain=True):
    if istrain:
        TFRecord_path = TFRecord_Train_Path
    else:
        TFRecord_path = TFRecord_Test_Path
    sim, l0, v0, l1, v1 = read_TFRecord(TFRecord_path)
    sim_batch, l0_batch, v0_batch, l1_batch, v1_batch = \
        tf.compat.v1.train.shuffle_batch([sim, l0, v0, l1, v1],
                                         batch_size=num, num_threads=2, capacity=100, min_after_dequeue=50)
    return sim_batch, l0_batch, v0_batch, l1_batch, v1_batch
```

get 函数的参数 num，就是每次取几个数据，istrain 选择获取的是训练集数据还是测试集数据。read_TFRecord 函数是辅助函数，用来解析 tfrecord 中的每一个 example。

2. 指数衰减学习率：

随着训练的次数增加，损失函数逐渐到达极小值（可能是局部最小值），这时候如果学习率太大，会出现震荡，所以需要减少学习率,利用tensorflow中的指数衰减函数来动态调整学习率。函数定义如下：

$$exponential\_decay(learning\_rate, global\_step\ , decay\_steps, decay\_rate, staircase = Flase, name = None)$$

衰减公式如下：

$$learning\_rate = learning\_rate * decay\_rate^{\frac{global\_step}{decay\_step}}$$

3. 滑动平均值：

利用参数的滑动平均值来增强模型的泛化能力，防止过拟合。可以用tensorflow中的ExponentialMovingAverage函数实现，函数定义如下：

$$ExponentialMovingAverage(decay, step)$$

滑动平均值的计算公式如下：

影子 = 衰减率 * 影子 + (1-衰减率) * 参数

五. 参考资料

1. https://www.cnblogs.com/bonelee/p/10475453.html
2. https://zhuanlan.zhihu.com/p/28749444
3. https://blog.csdn.net/coderTC/article/details/73864097
4. https://blog.csdn.net/luoxuexiong/article/details/90345143
5. Tang Shancheng ; Bai Yunyue ; Ma Fuyu. A Semantic Text Similarity Model for Double Short Chinese Sequences. IEEE, 2018