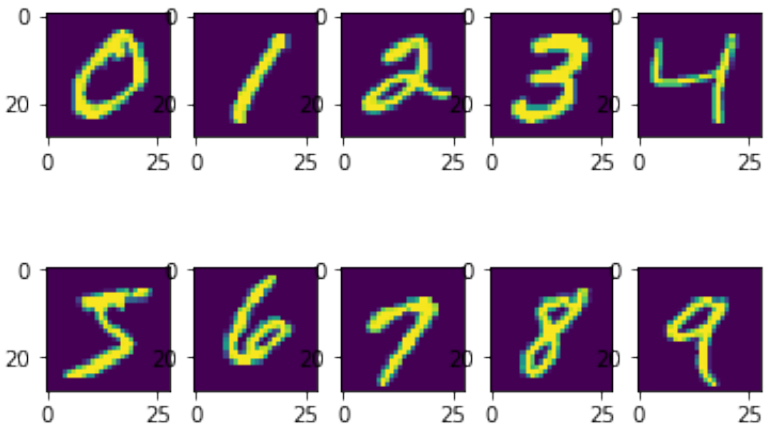
加载数据

In [7]:
```
from keras.datasets import mnist
(X0,Y0),(X1,Y1) = mnist.load_data(path="/clubear/datasets/mnist.npz")
print(X0.shape)
from matplotlib import pyplot as plt
plt.figure()
fig,ax = plt.subplots(2,5)
ax=ax.flatten()
for i in range(10):
    Im=X0[Y0==i][0]
    ax[i].imshow(Im)
plt.show()
```

```
(60000, 28, 28)

<Figure size 432x288 with 0 Axes>
```



数据处理

In [9]:
```
from keras.utils import np_utils
#X0不能直接被Tensorflow处理因为缺少一个argument：通道数
N0=X0.shape[0];N1=X1.shape[0]
print([N0,N1])
X0 = X0.reshape(N0,28,28,1)/255
X1 = X1.reshape(N1,28,28,1)/255
#变成one_hot变量
YY0 = np_utils.to_categorical(Y0)
YY1 = np_utils.to_categorical(Y1)
YY1
```

```
[60000, 10000]
```

Out[9]:
```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

LeNet5 模型搭建

In [10]:
```
from keras.layers import Conv2D,Dense,Flatten,Input,MaxPooling2D
,Dropout
from keras import Model

input_layer = Input([28,28,1])
x = input_layer
x = Conv2D(6,[5,5],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)
x = Conv2D(16,[5,5],padding = "valid", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)
x = Flatten()(x)
x = Dense(120,activation = 'relu')(x)
x = Dense(84,activation = 'relu')(x)
x = Dense(10,activation = 'softmax')(x)
output_layer=x
model=Model(input_layer,output_layer)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_5 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_6 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

模型和参数解释

第一层input layer，规定input只能是 [28,28,1] 的矩阵，无参数。
第二层conv2D： （5 *5* 1+1） *6 = 26* 6 = 156, 其中5 *5是kernel size，1是上一层遗留通道数，6是卷积核个数。*
第三层池化使用 [2,2] 矩形，步长行列都是2 （没有1 1tensor被重复池化），没有学习项，无参数
第四层conv2D: （5 *5* 6 + 1） *16 = 129* 16 = 2416, 其中5 *5是kernel size，6是上一层遗留通道数，16是卷积核个数。*
第五层池化使用 [2,2] 矩形，步长行列都是2 （没有1 1 tensor被重复池化），没有学习项，无参数

第六层压扁，无参数

第七层Dense： （hidden） 400 *120 + 120 = 48000 +120 = 48120*
第八层Dense： (hidden） 120 84 + 84 = 10080 +84 = 10164
第九层Dense： 84 * 10 + 10 = 840 +10 = 850

所以总共有61706个参数。

LeNet5 编译运行

In [11]:
```
model.compile(loss = 'categorical_crossentropy',optimizer='adam'
,metrics = ['accuracy'])
model.fit(X0,YY0,epochs = 10,batch_size = 200,validation_data=[X
1,YY1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 2s 30us/step - l
oss: 0.3687 - accuracy: 0.8902 - val_loss: 0.1072 - val_accurac
y: 0.9684
Epoch 2/10
60000/60000 [==============================] - 2s 25us/step - l
oss: 0.1042 - accuracy: 0.9684 - val_loss: 0.0660 - val_accurac
y: 0.9781
Epoch 3/10
60000/60000 [==============================] - 2s 26us/step - l
oss: 0.0739 - accuracy: 0.9772 - val_loss: 0.0771 - val_accurac
y: 0.9766
Epoch 4/10
60000/60000 [==============================] - 2s 25us/step - l
oss: 0.0589 - accuracy: 0.9827 - val_loss: 0.0464 - val_accurac
y: 0.9852
Epoch 5/10
60000/60000 [==============================] - 2s 25us/step - l
oss: 0.0482 - accuracy: 0.9849 - val_loss: 0.0424 - val_accurac
y: 0.9868
Epoch 6/10
60000/60000 [==============================] - 2s 26us/step - l
oss: 0.0400 - accuracy: 0.9875 - val_loss: 0.0381 - val_accurac
y: 0.9879
Epoch 7/10
60000/60000 [==============================] - 2s 27us/step - l
oss: 0.0362 - accuracy: 0.9882 - val_loss: 0.0340 - val_accurac
y: 0.9893
Epoch 8/10
60000/60000 [==============================] - 2s 27us/step - l
oss: 0.0297 - accuracy: 0.9902 - val_loss: 0.0432 - val_accurac
y: 0.9852
Epoch 9/10
60000/60000 [==============================] - 2s 26us/step - l
oss: 0.0281 - accuracy: 0.9908 - val_loss: 0.0305 - val_accurac
y: 0.9905
Epoch 10/10
60000/60000 [==============================] - 2s 27us/step - l
oss: 0.0248 - accuracy: 0.9917 - val_loss: 0.0375 - val_accurac
y: 0.9883
```

Out[11]:
```
<keras.callbacks.callbacks.History at 0x7f48702093d0>
```

Accuracy 高达0.9917!

改进？变形?

1. 通过google得知原先paper里的LeNet用的是Tanh作为activation function 而不是Relu，而后期人们发现 Relu会有更高的分类精度(对于MNIST)并且更简单。

1. 还可以考虑的变量：parameters

    - 卷积核大小
    - 卷积核个数
    - 池化规格
    - same padding or valid padding
    - optimal number of parameters, hidden layers etc...

1. Dropout?
   通过阅读资料，Dropout 适用于避免过度拟合（Overfitting），而LeNet高达99.17%的accuracy说明并不存在这一问题，Dropout不必要。