加载MNIST数据

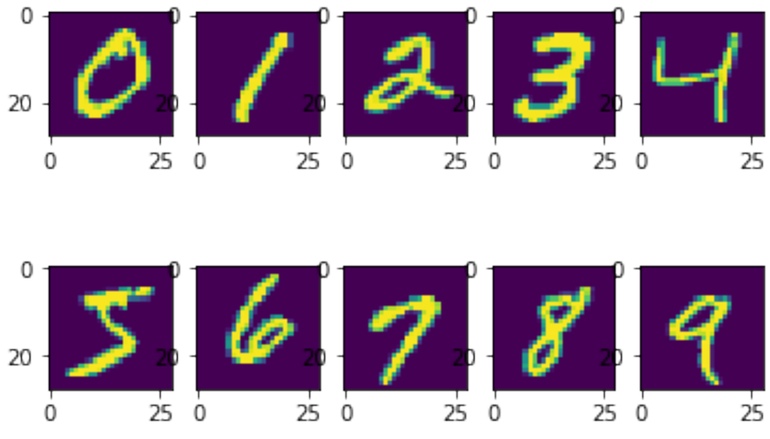
In [1]:
```
import keras
(X0, Y0), (X1, Y1) = keras.datasets.mnist.load_data()
```

```
Using TensorFlow backend.

Downloading data from https://s3.amazonaws.com/img-datasets/mni
st.npz
11493376/11490434 [==============================] - 1824s 159u
s/step
```

数据展示

In [2]:
```
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]      #Im = Y0对应的数字（eg 1）对应的图片（1的）里
的第一张
    ax[i].imshow(Im)
plt.show()
```

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



产生One-Hot型因变量

In [3]:
```
from keras.utils import to_categorical
YY0=to_categorical(Y0)
YY1=to_categorical(Y1)
YY1
#TensorFlow只能read One-Hot类型的Y
```

Out[3]:
```
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)
```

逻辑回归模型

In [4]:
```
from keras.layers import Activation, Dense, Flatten, Input
from keras import Model

input_shape=(28,28) #一张图片的size
input_layer=Input(input_shape)
x=input_layer
x=Flatten()(x) #使图片变成1*784的矩阵
x=Dense(10)(x) #全链接层，最后output要是10以对应10个digits
x=Activation('softmax')(x) #把output变成probability
output_layer=x
model=Model(input_layer,output_layer)
```

In [5]:
```
model.summary() #查看模型参数
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 28, 28)            0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                7850      
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0         
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

Dense_1 有 784 *10 （= 7840） + 10 = 7850 个参数 （weights）。因为在全连接层中每一个从 flatten_1来的都要连接到10个nodes，所以784* 10； 剩下的10个weights来自截距项

In [6]:
```
from keras.optimizers import Adam
model.compile(optimizer = Adam(0.01),
              loss = 'categorical_crossentropy',#LR比较常用的的loss function
              metrics = ['accuracy'])
```

In [7]:
```
model.fit(X0,YY0,
          validation_data=(X1,YY1),
          batch_size=1000,
          epochs=20)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 1s 15us/step - loss: 27.4137 - accuracy: 0.8112 - val_loss: 9.9834 - val_accuracy: 0.9027
Epoch 2/20
60000/60000 [==============================] - 0s 6us/step - loss: 7.7616 - accuracy: 0.9006 - val_loss: 7.0122 - val_accuracy: 0.8939
Epoch 3/20
60000/60000 [==============================] - 0s 6us/step - lo
```

```
ss: 6.2137 - accuracy: 0.8925 - val_loss: 5.8425 - val_accuracy
: 0.8942
Epoch 4/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.3354 - accuracy: 0.8907 - val_loss: 5.6379 - val_accuracy
: 0.8954
Epoch 5/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.1785 - accuracy: 0.8890 - val_loss: 5.6164 - val_accuracy
: 0.8858
Epoch 6/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.0149 - accuracy: 0.8892 - val_loss: 6.9368 - val_accuracy
: 0.8712
Epoch 7/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.4741 - accuracy: 0.8859 - val_loss: 5.2232 - val_accuracy
: 0.9061
Epoch 8/20
60000/60000 [==============================] - 0s 5us/step - lo
ss: 5.5999 - accuracy: 0.8910 - val_loss: 5.7054 - val_accuracy
: 0.9019
Epoch 9/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.2817 - accuracy: 0.8910 - val_loss: 6.8126 - val_accuracy
: 0.8707
Epoch 10/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.4920 - accuracy: 0.8888 - val_loss: 6.5173 - val_accuracy
: 0.8966
Epoch 11/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.1358 - accuracy: 0.8943 - val_loss: 6.5194 - val_accuracy
: 0.8836
Epoch 12/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.4373 - accuracy: 0.8903 - val_loss: 6.5411 - val_accuracy
: 0.8889
Epoch 13/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 4.8872 - accuracy: 0.8974 - val_loss: 6.4614 - val_accuracy
: 0.8851
Epoch 14/20
60000/60000 [==============================] - 0s 5us/step - lo
ss: 4.8789 - accuracy: 0.8943 - val_loss: 7.3454 - val_accuracy
: 0.8722
Epoch 15/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.8168 - accuracy: 0.8894 - val_loss: 6.6594 - val_accuracy
: 0.8972
Epoch 16/20
60000/60000 [==============================] - 0s 5us/step - lo
ss: 5.4283 - accuracy: 0.8941 - val_loss: 6.8719 - val_accuracy
: 0.8802
Epoch 17/20
```

```
60000/60000 [==============================] - 0s 5us/step - lo
ss: 5.3954 - accuracy: 0.8921 - val_loss: 7.3211 - val_accuracy
: 0.8730
Epoch 18/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.5083 - accuracy: 0.8942 - val_loss: 6.6847 - val_accuracy
: 0.8836
Epoch 19/20
60000/60000 [==============================] - 0s 5us/step - lo
ss: 6.4071 - accuracy: 0.8862 - val_loss: 8.1961 - val_accuracy
: 0.8879
Epoch 20/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 5.5712 - accuracy: 0.8956 - val_loss: 9.4722 - val_accuracy
: 0.8706
```

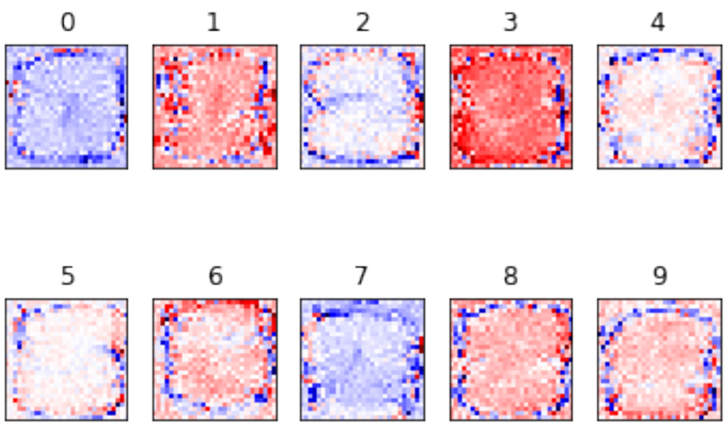
Out[7]: <keras.callbacks.callbacks.History at 0x7f783c22bd50>

得到accuracy在0.9左右

参数估计结果可视化

In [8]:

```
fig,ax = plt.subplots(2,5)
ax=ax.flatten()
weights = model.layers[2].get_weights()[0]
for i in range(10):
    Im=weights[:,i].reshape((28,28))
    ax[i].imshow(Im,cmap='seismic')
    ax[i].set_title("{}".format(i))
    ax[i].set_xticks([])
    ax[i].set_yticks([])
plt.show()
```



好像并没能看出digits间特别大的差异，但是有一些轮廓可以大概看清。

改进模型

In [9]:
```
input_shape=(28,28) #一张图片的size
input_layer=Input(input_shape)
x=input_layer
x=Flatten()(x) #使图片变成1*784的矩阵
x=Dense(70)(x) #全连接层，目的是为了先根据一些digits间的common featur
e进行简单分类
x=Dense(10)(x) #全连接层，最后output要是10以对应10个digits
x=Activation('softmax')(x) #把output变成probability
output_layer=x
model1=Model(input_layer,output_layer)
```

In [10]:
```
model1.compile(optimizer = Adam(0.01),
               loss = 'categorical_crossentropy',#LR比较常用的的loss
function
               metrics = ['accuracy'])
```

In [11]:
```
model1.fit(X0,YY0,
          validation_data=(X1,YY1),
          batch_size=1000,
          epochs=20)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 1s 9us/step - lo
ss: 86.3689 - accuracy: 0.7965 - val_loss: 19.4118 - val_accura
cy: 0.8983
Epoch 2/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 11.9134 - accuracy: 0.9011 - val_loss: 8.4536 - val_accurac
y: 0.8981
Epoch 3/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 6.1554 - accuracy: 0.8946 - val_loss: 6.2628 - val_accuracy
: 0.8694
Epoch 4/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 4.1176 - accuracy: 0.8895 - val_loss: 4.2979 - val_accuracy
: 0.8793
Epoch 5/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 3.2036 - accuracy: 0.8908 - val_loss: 3.2912 - val_accuracy
: 0.8733
Epoch 6/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 2.3869 - accuracy: 0.8894 - val_loss: 2.2883 - val_accuracy
: 0.8946
Epoch 7/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 1.7733 - accuracy: 0.8926 - val_loss: 2.0455 - val_accuracy
```

```
: 0.8887
Epoch 8/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 1.4598 - accuracy: 0.8906 - val_loss: 1.4061 - val_accuracy
: 0.8962
Epoch 9/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 1.1061 - accuracy: 0.8931 - val_loss: 1.2074 - val_accuracy
: 0.8827
Epoch 10/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 1.0735 - accuracy: 0.8858 - val_loss: 1.2105 - val_accuracy
: 0.8816
Epoch 11/20
60000/60000 [==============================] - 0s 7us/step - lo
ss: 0.8356 - accuracy: 0.8917 - val_loss: 0.8572 - val_accuracy
: 0.8839
Epoch 12/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.6454 - accuracy: 0.8941 - val_loss: 0.9397 - val_accuracy
: 0.8774
Epoch 13/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.5621 - accuracy: 0.8995 - val_loss: 0.6272 - val_accuracy
: 0.8910
Epoch 14/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.4820 - accuracy: 0.9003 - val_loss: 0.5183 - val_accuracy
: 0.9039
Epoch 15/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.4569 - accuracy: 0.9004 - val_loss: 0.5136 - val_accuracy
: 0.9008
Epoch 16/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.4213 - accuracy: 0.9036 - val_loss: 0.5154 - val_accuracy
: 0.8923
Epoch 17/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.4246 - accuracy: 0.9027 - val_loss: 0.4950 - val_accuracy
: 0.8998
Epoch 18/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.4054 - accuracy: 0.9043 - val_loss: 0.4534 - val_accuracy
: 0.9036
Epoch 19/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.3724 - accuracy: 0.9072 - val_loss: 0.4359 - val_accuracy
: 0.8981
Epoch 20/20
60000/60000 [==============================] - 0s 6us/step - lo
ss: 0.3763 - accuracy: 0.9051 - val_loss: 0.4625 - val_accuracy
: 0.8966
```

```
Out[11]: <keras.callbacks.callbacks.History at 0x7f783c3723d0>
```

可以看到accuracy依旧在9.0左右，但是从0.89 上升到了0.90

In [16]:
```
model1.summary()
```

```
Model: "model_2"

_________________________________________________________________
__
Layer (type)                 Output Shape              Param #
=================================================================
==
input_2 (InputLayer)         (None, 28, 28)            0

_________________________________________________________________
__
flatten_2 (Flatten)          (None, 784)               0

_________________________________________________________________
__
dense_2 (Dense)              (None, 70)                54950

_________________________________________________________________
__
dense_3 (Dense)              (None, 10)                710

_________________________________________________________________
__
activation_2 (Activation)    (None, 10)                0
=================================================================
==
Total params: 55,660
Trainable params: 55,660
Non-trainable params: 0

_________________________________________________________________
__
```

In [20]:
```
#dense_2 Param: (28 *28 =) 784  * 70  (= 54880) + 70 = 54950
#dense_3 Param: 70*10 + 10 = 710
```

In [13]:
```
model1.fit(X0,YY0,
          validation_data=(X1,YY1),
          batch_size=5000,
          epochs=20)
#更改了batch_size, 因为55000个data 一个batch只取1000有点少
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2635 - accuracy: 0.9260 - val_loss: 0.3451 - val_accuracy
: 0.9154
Epoch 2/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2419 - accuracy: 0.9326 - val_loss: 0.3220 - val_accuracy
: 0.9204
Epoch 3/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2347 - accuracy: 0.9353 - val_loss: 0.3164 - val_accuracy
```

```
: 0.9210
Epoch 4/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2328 - accuracy: 0.9352 - val_loss: 0.3240 - val_accuracy
: 0.9196
Epoch 5/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2348 - accuracy: 0.9341 - val_loss: 0.3219 - val_accuracy
: 0.9214
Epoch 6/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2349 - accuracy: 0.9344 - val_loss: 0.3212 - val_accuracy
: 0.9202
Epoch 7/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2324 - accuracy: 0.9347 - val_loss: 0.3236 - val_accuracy
: 0.9204
Epoch 8/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2331 - accuracy: 0.9349 - val_loss: 0.3220 - val_accuracy
: 0.9206
Epoch 9/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2322 - accuracy: 0.9355 - val_loss: 0.3220 - val_accuracy
: 0.9204
Epoch 10/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2321 - accuracy: 0.9355 - val_loss: 0.3199 - val_accuracy
: 0.9216
Epoch 11/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2314 - accuracy: 0.9355 - val_loss: 0.3195 - val_accuracy
: 0.9218
Epoch 12/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2333 - accuracy: 0.9351 - val_loss: 0.3203 - val_accuracy
: 0.9214
Epoch 13/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2311 - accuracy: 0.9357 - val_loss: 0.3203 - val_accuracy
: 0.9187
Epoch 14/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2315 - accuracy: 0.9351 - val_loss: 0.3151 - val_accuracy
: 0.9217
Epoch 15/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2306 - accuracy: 0.9352 - val_loss: 0.3201 - val_accuracy
: 0.9220
Epoch 16/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2326 - accuracy: 0.9348 - val_loss: 0.3277 - val_accuracy
: 0.9192
Epoch 17/20
60000/60000 [==============================] - 0s 3us/step - lo
```

```
ss: 0.2344 - accuracy: 0.9338 - val_loss: 0.3196 - val_accuracy
: 0.9214
Epoch 18/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2336 - accuracy: 0.9346 - val_loss: 0.3256 - val_accuracy
: 0.9183
Epoch 19/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2310 - accuracy: 0.9357 - val_loss: 0.3280 - val_accuracy
: 0.9188
Epoch 20/20
60000/60000 [==============================] - 0s 3us/step - lo
ss: 0.2342 - accuracy: 0.9345 - val_loss: 0.3261 - val_accuracy
: 0.9185
```

Out[13]:
```
<keras.callbacks.callbacks.History at 0x7f78300aaf50>
```

accruacy 上升到了0.93

更多基于图片的分类问题

应用一 动物分类
例子： 狗狗品种分类，Dog or muffin 等
（以 dog or muffin 举例）
X：一张要么是狗要么是 muffin 的图片
Y：狗或 muffin

应用二 植物分类
例子： 识别不同植物种类的 APP
X：对一个植物拍照 -> 提取有关植物的信息（叶片形状，茎长宽，颜色等）
Y：与 X 信息相匹配的植物名称

应用三 物品识别
例子：垃圾分类
X：一张垃圾的图片
Y：可回收物/其他垃圾/有害垃圾等垃圾的分类