

# 无人驾驶汽车系统入门（九）——神经网络基础

原创

置顶

AdamShan

2018-01-08 16:52:47

9784

收藏 36

版权

分类专栏:

无人驾驶汽车专题

无人驾驶汽车系统入门

文章标签:


无人驾驶

无人车

神经网络

深度学习

keras



自动驾驶系统进阶与项目实践

结合本人自动驾驶行业研发经验，从传感器数据融合、深度学习环境感知、高精度地图和定位、决策...

AdamShan

¥29.90

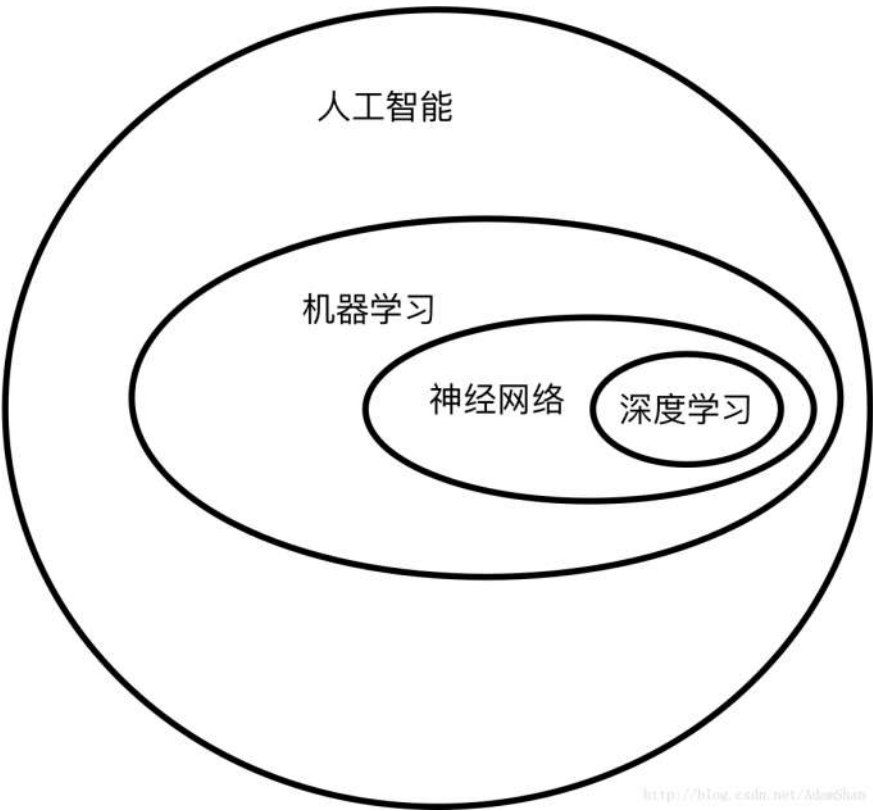
订阅博主

# 无人驾驶汽车系统入门（九）——神经网络基础

在上一节中，我们介绍了机器学习的相关基础，尤其是知道了监督学习的基本构成因素：数据，模型，策略和算法。在本节，我们具体学习一种监督学习算法——神经网络。现代深度学习模型其本质均为人工神经网络，所以在进一步探索深度学习在无人驾驶中的应用之前，我们先了解一下神经网络的理论基础和代码实现。

创作不易，转载请注明出处：<http://blog.csdn.net/adamshan/article/details/79004784>

既然说到神经网络和深度学习，那么我们就先来理一理我们前面介绍的各种概念的关系和范畴，如下图所示，是这些概念的定义范畴：



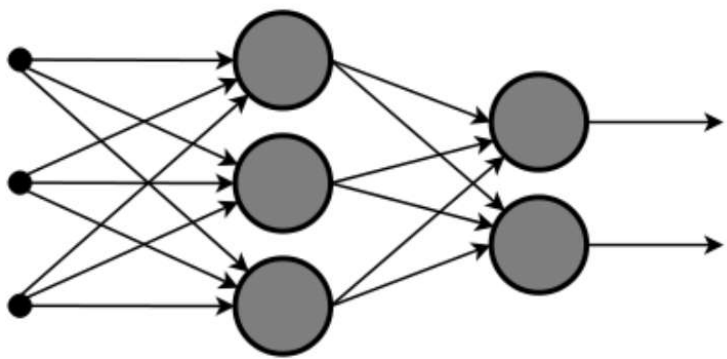
人工智能是一个很大的定义，机器学习是人们在探索人工智能的过程中的一种思路（并不一定是一条通往终极人工智能的路，所以人工智能绝对不等于机器学习！），神经网络是机器学习中的一种监督学习算法，而深度学习则是将神经网络的层数增多，使用大量数据来建立的一种表示学习算法（关于表示学习，我们会在深度学习一节详细论述）。所以我们就先从神经网络入手，再进一步学习深度学习。

我们以上一节介绍的监督学习的几个因素（模型，策略和算法）来逐一介绍神经网络算法中的这几个因素，任务我们还是沿用上一节介绍的手写数字识别任务，当然，数据就是MNIST数据了：

## 神经网络入门

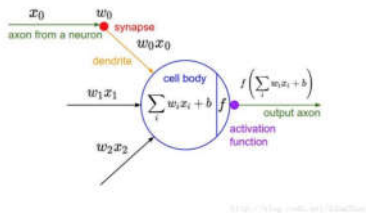
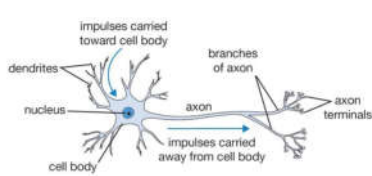
## 基本结构

神经网络的模型当然就是神经网络了，它的一个基本结构如下图所示：



<http://blog.csdn.net/AdamShan>

这是一个形象化的表示，它看起来就像我们人类的神经网络（这也是这个模型名称的由来），其中的每一个小圈圈，就像我们人类的一个神经元，一条条边就像神经元之间的突触（人的神经元通过突触相连接），我们取出神经网络中的一个神经元，它的结构如下图所示的右侧，这个人造的神经元我们称之为 **感知机 (Perceptron)**，下图的左侧则是人类的一个神经细胞：



<http://blog.csdn.net/AdamShan>

由上图可知，在神经网络中，一个神经元能够接受多个输入，我们用向量  $x = (x_0, x_1, \dots, x_n)$  来表示输入的数值，在神经元的输入边上，输入会乘以一个权重向量  $w = (w_0, w_1, \dots, w_n)$ ，在神经元中，这些乘以权重的输入被求和，并且加上了一个很小的偏置  $b$ ，所以神经元的值就变成了：

$$h = \sum_i w_i x_i + b$$

这里的  $w$  和  $b$  就是神经网络需要训练的参数。在神经元的输出端， $h$  被输入到一个 **激活函数(activation function)  $f$**  中，即：

$$output = f(h) = f\left(\sum_i w_i x_i + b\right)$$

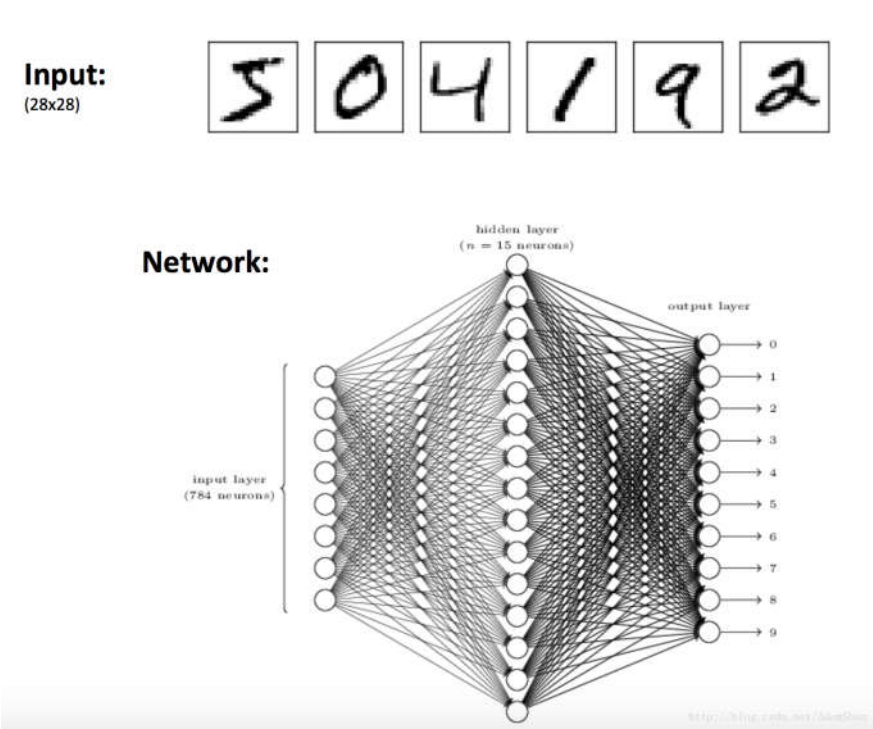
这里的激活函数通常是非线性函数，包括 *sigmoid* 函数，*tanh* 函数，*ReLU* ()等等，下图总结了目前常用的激活函数的图像，表达式以及导数：

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

那么我们的神经网络呢实际上就是由这么一些神经元通过连接组合而成。前一个神经元的尾部（输出）连接到后一个神经元的的一个输入，这样便形成了一个层次状的机构，那么在整个神经网络的第一层，我们称之为 **输入层(input layer)**，它的输入数值便是我们的数据，最后一层我们称之为 **输出层(output layer)**，中间的若干层被称为 **隐含层 (hidden layer)**。这就是一个神经网络的基本结构，们讨论模型时强调我们的模型应当具有一定的容量，这个容量表现在能够拟合我们要解决的任务的函数。也就是说我们的模型应该具有拟合我们想要的函数的能力。神经网络的强大之处就在于，**通过改变神经网络的结构和参数规模，神经网络能够拟合任意函数！**，换句话说，神经网络拥有无限容量。

### 前向传播

在了解神经网络的无限容量以后，开始使用它来拟合我们要解决的机器学习任务——手写字识别，在这个问题中，手写字的图片均为  $28 \times 28$ ，所以输入的向量X长度为 784 ,手写字一共有10个类别， 分别是0到9，假设我们使用如下图的神经网络来处理这个任务：



那么输出层的每一个神经元的输出对应一个类别，所以输出层是10个神经元，实际上，我们只需要4个输出就能表示0-9了，但是在实践中使用对应类别数的输出能够表现出更好的性能，这种单点激活的编码方式我们称为one-hot编码，即对应类别的位置激活为1，其他位置为0。这种编码方式在类别数很少的情况下是一种非常有效的方式。

神经网络在训练的过程中输出的是各个点的“得分”，这个数值，被称为 **logit**，我们希望得到的不是得分而是输出层各个神经元被激活的概率，所以我们在神经网络的输出层在添加一个 *Softmax* 层，它的作用就是对输出层每个神经元的输出值计算一个概率，概率和为1,输出的值越大的点概率也就越大。

Softmax函数能将一个含任意实数的K维的向量  $z$  的“压缩”到另一个K维实向量  $\sigma(z)$  中，使得每一个元素的范围都在  $(0, 1)$  之间，并且所有元素的和为1。计算公式如下：

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad j = 1, \dots, K.$$

为了更加直观，我们使用Python实现一个SoftMax函数：

```
1 import numpy as np
2
3 def softmax(x):
4     """Compute softmax values for each sets of scores in x."""
5     e_x = np.exp(x - np.max(x))
6     return e_x / e_x.sum()
7
8 scores = [3.0, 1.0, 0.2]
9 print(softmax(scores))
```

得到的输出结果为：

```
1 [ 0.8360188  0.11314284  0.05083836]
```

有个结果的概率之后,我们计算损失函数，对于具体的任务，要定义相应的损失函数，对于识别和分类这类多分类任务而言，最常用的损失函数就是 **交叉熵(Cross Entropy)**，它的表达式如下：

$$L(\theta) = -\frac{1}{n} \sum_n [y \ln a + (1 - y) \ln(1 - a)]$$

其中  $y$  是真实的标签， $a$  则是神经元的输出。这样的损失函数具有两个很好的性质：

- 非负性：这样我们就可以最小化损失函数了
- 真实值与模型的输出值接近时损失函数的值也趋向于0

那么在求得损失以后，也就得到了神经网络的输出结果（对于手写字识别而言，就是识别的结果）以及输出结果与真实值之间的“距离”。我们把这么一个过程称为一次 **前向传播 (forward propagation)**，显然，前向传播能够输出正确的分类的前提是神经网络已经具有了合适的参数，那么如何调整合适的参数呢？神经网络通过 **反向传播 (back propagation)** 算法让来自损失函数的信息通过网络向后流动，从而计算梯度信息。反向传播这个术语通常被人误解为神经网络的整个学习算法，**实际上它只是一种计算梯度的方法**，而 **随机梯度下降 (stochastic gradient descent, SGD)** 才是使用梯度进行学习的算法，通过随机梯度下降算法，深度网络能够自行完成参数的调整，最小化损失函数，即最小化经验风险，从未使得模型的变现近似于我们要处理的任务的模式。

## 随机梯度下降

随机梯度下降是几乎所有的深度学习模型都采用的一种学习算法，在深度学习中，好的模型往往需要大量的数据，大量的数据带来的计算量也是巨大的，根据前一节我们知道，机器学习算法中的损失函数是通过计算每个样本的损失函数的和来求得的，那么训练数据的损失就可以写作：

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n L(x_i)$$

👍 点赞12

💬 评论5

🔗 分享

🌟 收藏36

💰 打赏

🚩 举报

订阅博主

那么对与这个相加的函数，梯度下降就需要计算：

$$\nabla_{\theta} L(\theta) = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L(x_i, y_i, \theta)$$

即使用整个样本集去求解梯度，对于大数据集合（数以百万，千万计的样本），这种做法是很没有效率的。

在随机梯度下降中，梯度被认为是一个期望，那么期望就可以通过小批量的样本来近似，因此，在执行梯度下降的过程中，我们不使用整个样本集来计算梯度，而是每次都随机取一个**小批量（minibatch）**的样本，这个小批量的样本数往往在一千以内（通常，我们习惯使用2的指数次最为小批量的样本个数，比如说64,128,256），假设小批量眼本数为  $m$ ，那么每次执行梯度下降，我们只需要将随机抽取的  $m$  个样本放入神经网络，经过前向传播，求解了损失，然后基于损失函数，反向求解这  $m$  个样本的梯度，这  $m$  个样本的梯度就可以看作整个样本集的梯度的估计：

$$g = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(x_i, y_i, \theta)$$

然后，再使用这个梯度的估计量来更新参数：

$$\theta \leftarrow \theta - \epsilon g$$

其中， $\epsilon$  是学习率。

随机梯度下降是在大规模数据上训练大型线性模型的主要方法。在深度学习兴起之前，学习非线性模型的主要方法是结合核技巧的线性模型。当数据集的样本数巨大时，这类方法的计算量是不能接受的。在学术界，深度学习从 2006 年开始收到关注的原因是，在数以万计样本的中等规模数据集上，深度学习在新样本上比当时很多热门算法泛化得更好。不久后，深度学习在工业界受到了更多的关注，因为其提供了一种训练大数据集上的非线性模型的可扩展方式。

## 使用keras实现神经网络

下面我们使用Keras快速实现一个三层的神经网络，网络的结构如上图所示，(784, 15, 10)，我们使用交叉熵作为损失函数，使用随机梯度下降作为模型的优化算法对模型进行训练。在训练完成以后，我们将这个模型保存成“model.json”文件，然后使用我们自己的手写字来验证一下模型的准确度。我们在jupyter notebook中逐步完成模型的训练，调整和验证。

## 数据准备

我们需要下载MNIST数据集到本地，然后将数据集读取到内存，并且切分为训练集和测试集。

引入我们需要的库，并且定义超参数：

```
1 from __future__ import print_function
2
3 import keras
4 from keras.datasets import mnist
5 from keras.models import Sequential
6 from keras.layers import Dense
7 from keras.optimizers import SGD
8 from matplotlib import pyplot as plt
9
10 batch_size = 128
11 num_classes = 10
12 epochs = 20
```

下载MNIST数据集，读取数据集，并打印数据集的大小：

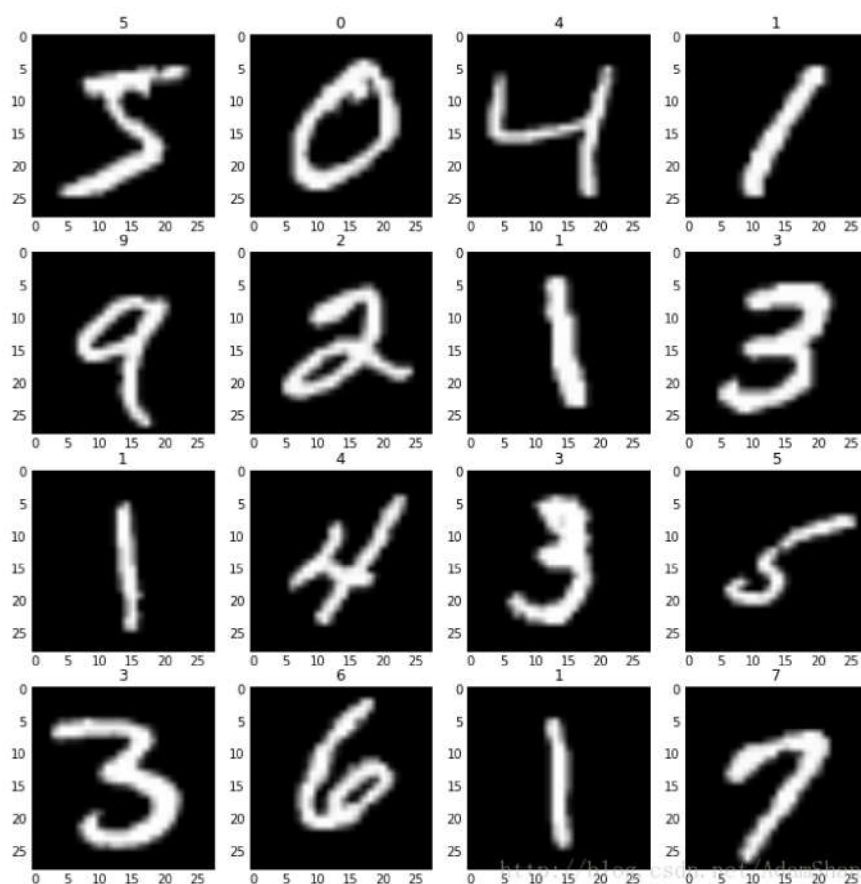
```
1 # the data, shuffled and split between train and test sets
2 (x_train, y_train), (x_test, y_test) = mnist.load_data()
3
4 print(x_train.shape, x_test.shape)
5 print(y_train.shape, y_test.shape)
```

结果:

```
1 (60000, 28, 28) (10000, 28, 28)
2 (60000,) (10000,)
```

可以看出, 这个数据集的训练集有6万个样本, 测试集有1万个样本, 每个样本是一张  $28 \times 28$  的图像, 我们用pyplot把这些图片展示一部分:

```
1 def show_samples(samples, labels):
2     """
3     display 16 samples and labels
4     """
5     plt.figure(figsize=(12, 12))
6     for i in range(len(samples)):
7         plt.subplot(4, 4, i+1)
8         plt.imshow(samples[i], cmap='gray')
9         plt.title(labels[i])
10    plt.show()
11
12 show_samples(x_train[:16], y_train[:16])
```



如上图所示, 展示了MNIST数据集的训练集的前16个样本图片和标签, 说明我们的数据读取正确, 由于我们的神经网络接受的是784这样一个维度的输入, 所以我们要把样本的形状做一下调整, 同时我们对标签进行一个 one-hot 编码:

```
1 x_train = x_train.reshape(60000, 784)
2 x_test = x_test.reshape(10000, 784)
3 x_train = x_train.astype('float32')
4 x_test = x_test.astype('float32')
5 # 将样本归一化
6 x_train /= 255
7 x_test /= 255
8
9 # convert class vectors to binary class matrices
10 y_train = keras.utils.to_categorical(y_train, 10)
```



```

11 y_test = keras.utils.to_categorical(y_test, num_classes)
12 print(x_train.shape, x_test.shape)
13 print(y_train.shape, y_test.shape)

```

```

1 (60000, 784) (10000, 784)
2 (60000, 10) (10000, 10)

```

接下来我们构造神经网络，我们先仅使用15个隐含层神经元看一下训练的效果：

```

1 model = Sequential()
2 model.add(Dense(15, activation='relu', input_shape=(784,)))
3 model.add(Dense(num_classes, activation='softmax'))
4
5 model.summary()
6
7 model.compile(loss='categorical_crossentropy',
8               optimizer=SGD(lr=0.01),
9               metrics=['accuracy'])
10
11 history = model.fit(x_train, y_train,
12                    batch_size=batch_size,
13                    epochs=epochs,
14                    verbose=1,
15                    validation_data=(x_test, y_test))
16 ### print the keys contained in the history object
17 print(history.history.keys())

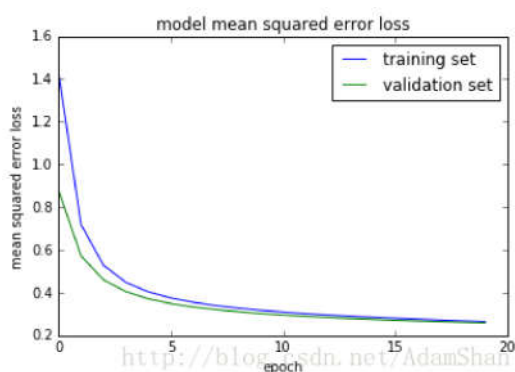
```

画出训练过程的损失情况：

```

1 def plot_training(history):
2     ### plot the training and validation loss for each epoch
3     plt.plot(history.history['loss'])
4     plt.plot(history.history['val_loss'])
5     plt.title('model mean squared error loss')
6     plt.ylabel('mean squared error loss')
7     plt.xlabel('epoch')
8     plt.legend(['training set', 'validation set'], loc='upper right')
9     plt.show()
10
11 plot_training(history=history)

```



我们发现训练集和验证集的损失都是呈现一个下降的趋势，并且大约在20个epoch之后区域稳定。

我们使用测试集来验证一下模型识别的精度：

```

1 score = model.evaluate(x_test, y_test, verbose=0)
2 print('Test loss:', score[0])
3 print('Test accuracy:', score[1])

```

结果：

👍 点赞12

💬 评论5

🔗 分享

★ 收藏36

💰 打赏

🚩 举报

📄 订阅博主

```
1 Test loss: 0.2586659453
2 Test accuracy: 0.9249
```

发现我们的模型在测试集上的识别进度为92%，看起来很高，但是我们还是不满意，我们想改进我们的模型来提高识别的精度，其实，传统的BP神经网络就到此为止了，我们使用当前大热的深度神经网络的思维来改变一下代码，读者可能会对后面的代码产生困惑，但是不用担心，相关的理论知识我会在后面的博客中和大家一起详细讨论。

### 3层网络的一点小变动——深度前馈神经网络：

首先我们把我们原来的网络的层数进一步加深，变成2个隐含层，同时，我们将每一个隐含层的神经元数量扩大到512个，在每一个隐含层的后面，我们使用一种叫做 Dropout 的正则化的技术，最后，我们使用一种SGD的变体——RMSprop算法作为模型的学习算法，这样，我们的第一个深度神经网络就构造好了，来看看效果吧~

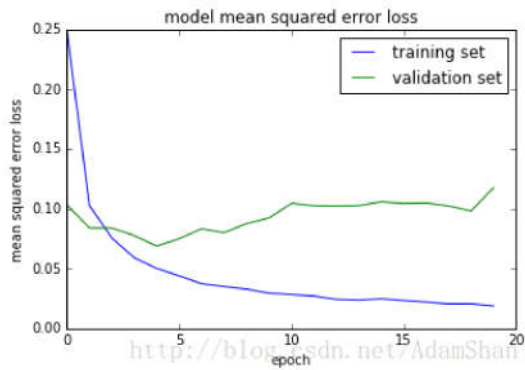
```
1 from keras.layers import Dropout
2 from keras.optimizers import RMSprop
3
4 model = Sequential()
5 model.add(Dense(512, activation='relu', input_shape=(784,)))
6 model.add(Dropout(0.2))
7 model.add(Dense(512, activation='relu'))
8 model.add(Dropout(0.2))
9 model.add(Dense(num_classes, activation='softmax'))
10
11 model.summary()
12
13 model.compile(loss='categorical_crossentropy',
14               optimizer=RMSprop(),
15               metrics=['accuracy'])
16
17 history = model.fit(x_train, y_train,
18                    batch_size=batch_size,
19                    epochs=epochs,
20                    verbose=1,
21                    validation_data=(x_test, y_test))
22
23 ### print the keys contained in the history object
24 print(history.history.keys())
25 plot_training(history=history)
26 model.save('model.json')
27
28 score = model.evaluate(x_test, y_test, verbose=0)
29 print('Test loss:', score[0])
30 print('Test accuracy:', score[1])
```

网络的结构和参数数量：

```
1 -----
2 Layer (type)          Output Shape          Param #
3 -----
4 dense_3 (Dense)        (None, 512)           401920
5 -----
6 dropout_1 (Dropout)    (None, 512)           0
7 -----
8 dense_4 (Dense)        (None, 512)           262656
9 -----
10 dropout_2 (Dropout)    (None, 512)           0
11 -----
12 dense_5 (Dense)        (None, 10)            5130
13 -----
14 Total params: 669,706
15 Trainable params: 669,706
16 Non-trainable params: 0
```

训练过程中的损失的变化：





测试集的识别精度:

```
1 Test loss: 0.117383948493
2 Test accuracy: 0.9824
```

我们的“深度”网络取得了 98% 的分类精度，比我们之前的3层网络确实好了很多，那么我们抽取测试集的16张图片来看看模型识别的效果吧：

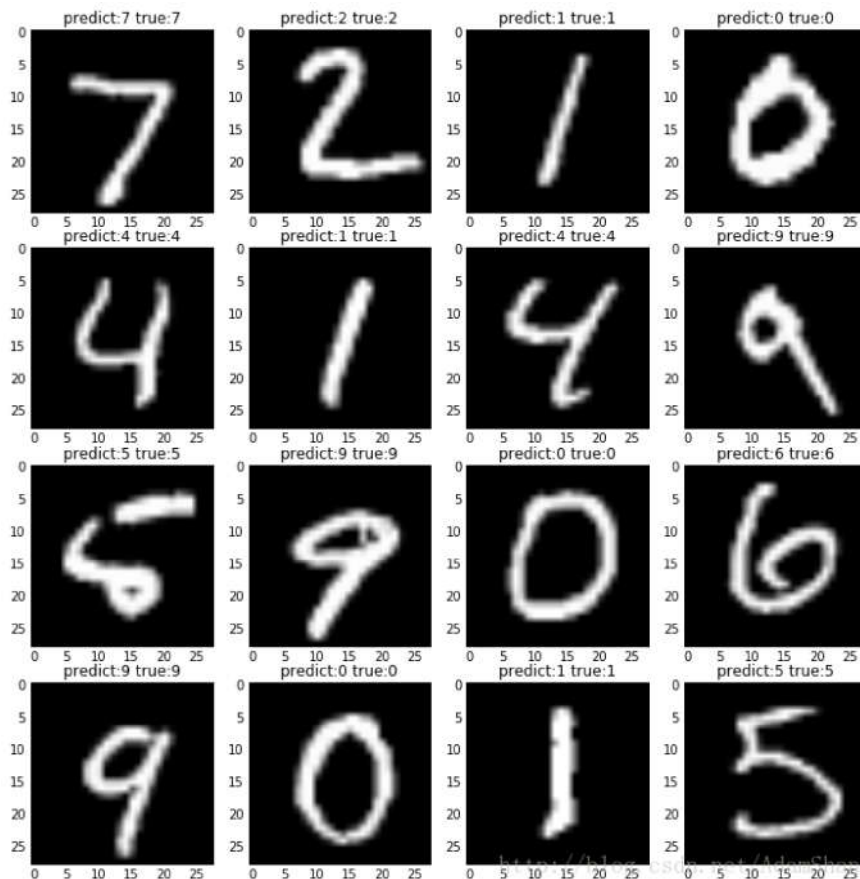
```
1 import numpy as np
2 result = model.predict(x_test[:16])
3 result = np.argmax(result, 1)
4 print('predict: ', result)
5 true = np.argmax(y_test[:16], 1)
6 print('true: ', true)
```

```
1 predict:  [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5]
2 true:    [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5]
```

看看实际的图片：

```
1 fig2 = plt.figure(figsize=(12, 12))
2 for i in range(16):
3     plt.subplot(4, 4, i+1)
4     plt.imshow(x_test[i].reshape((28, 28)), cmap='gray')
5     plt.title('predict:'+str(result[i])+' true:'+str(true[i]))
6 plt.show()
```



## 小结

至此，神经网络的基础部分我们就已经了解完了，下面我们会进一步了解深度学习的一些理论和实践，在接下来，我们将学习深度学习在无人驾驶的感知模块的应用，以及单纯基于深度学习的端到端无人驾驶技术。我们还会进一步探索强化学习+深度神经网络，学习应用于无人驾驶的强化学习理论和技术。

完整代码见链接：<http://download.csdn.net/download/adamshan/10194745>



AdamShan

1 专家

图像处理

深度学习

TensorFlow

奔驰高级自动驾驶扫地僧，谷歌认证机器学习专家，兰州大学无人驾驶团队创始人，主攻深度学习，无人驾驶汽车方向，著有《无人驾驶原理与实践》一书。

### 无人驾驶入门—Autware使用手册

04-03

无人驾驶汽车系统入门 - Autware\_UsersManual\_v1.1 Autware-用于城市自主驾驶的集成开源软件，...



优质评论可以帮助作者获得更高权重



评论



贰叁z: 为什么我看不公式。。。 5月前 回复 ...



Walliam\_Wu 回复: 我也是太难顶了 5月前 回复 ...



sjh\_sjh\_sjh: 训练集和验证集，测试集出现错误 7月前 回复 ...



sjh\_sjh\_sjh: 还有一点，文中的训练集60000张，测试集合10000张，代码居然将测试集直接当作验证集了，应该在训练集中分出一部分充当验证集 7月前 回复 ...



sjh\_sjh\_sjh: 为什么代码中误差使用的交叉熵误差，图像绘制的又是 均方误差呢？ 7月前 回复 ...



## 相关推荐

点赞12

评论5

分享

收藏36

打赏

举报

订阅博主