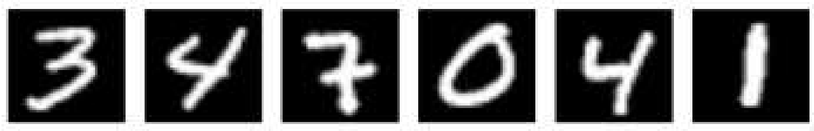
Neural Network Based on Numpy

Dataset

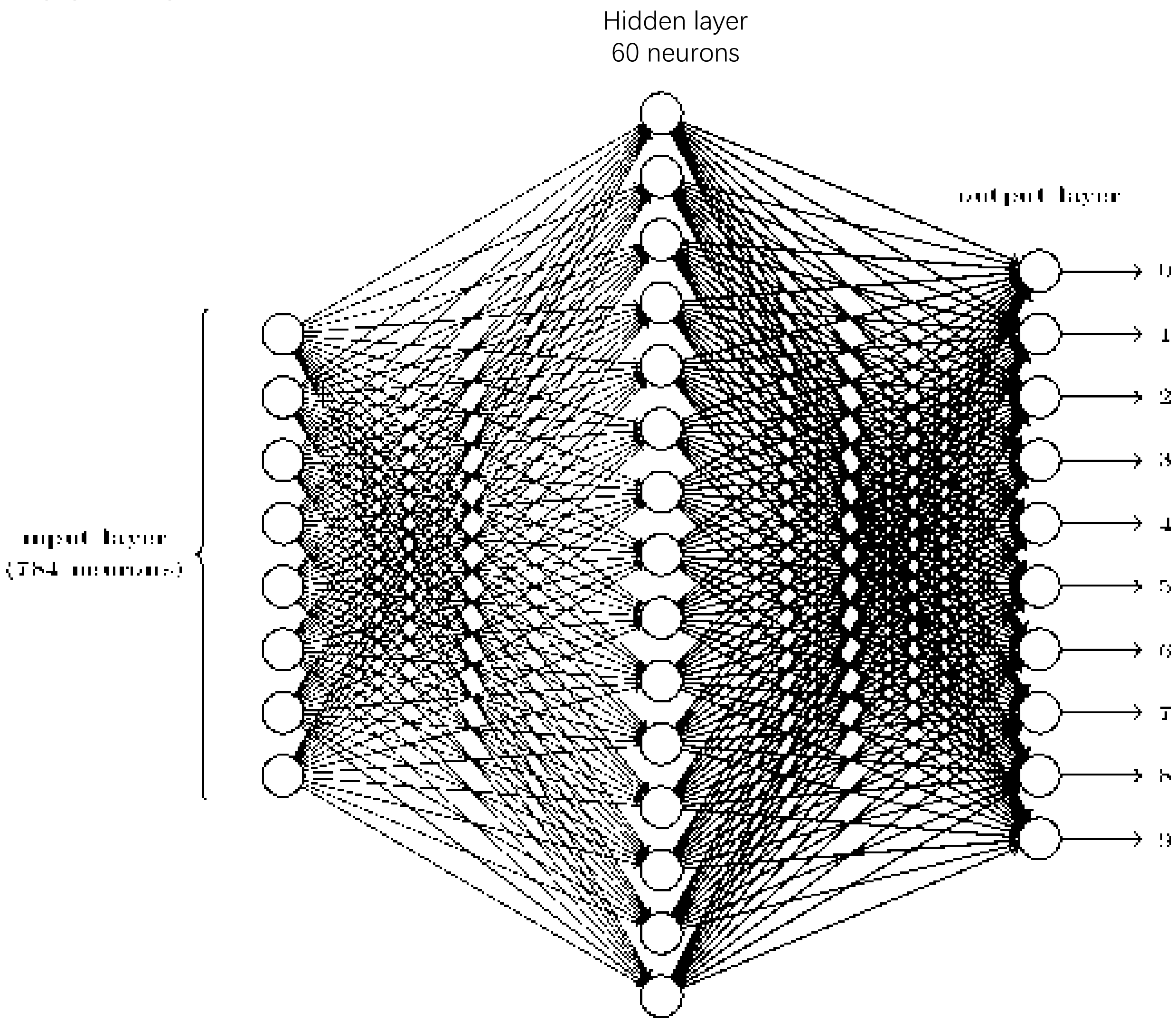
MNIST里包含各种手写数字图片以及每张图片对应的标签。每张图片都经过了大小归一化和居中处理。处理后的数据是一个单通道的黑白图片。



MNIST数据集中的图片是28X28Pixe，压缩为一维之后是$28 \times 28 = 784$

训练集：50000
验证集：10000
测试集：10000

Model Architecture

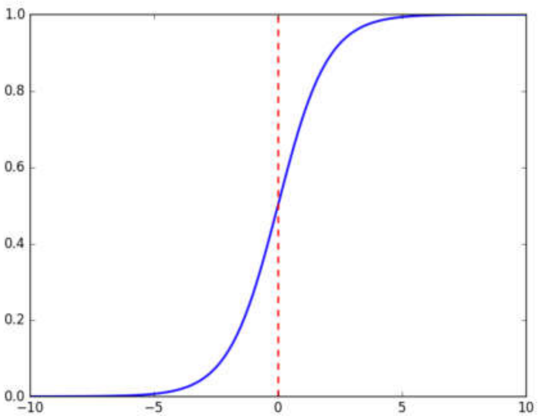


前向传播

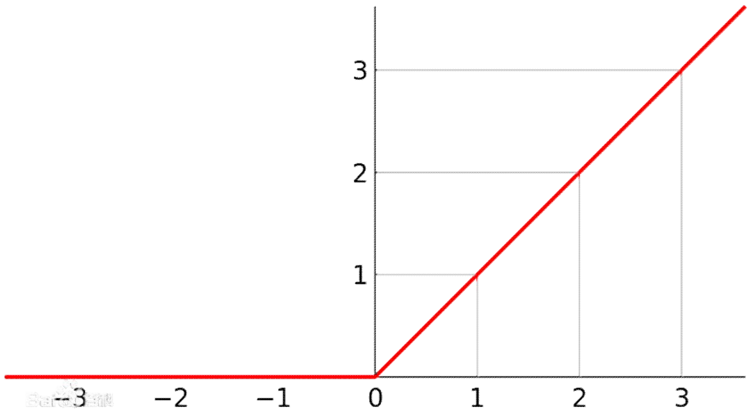
$w1 \in R^{60\times784}, x \in R^{784\times batch\ size}, b \in R^{60\times1},$
$z1 \in R^{60\times batch\ size},\ \ a1 \in\ R^{60\times batch\ size}$

$x \rightarrow z1 = w1x + b1 \rightarrow a1 = relu(z1) \rightarrow z2 = w2a1 + b2 \rightarrow a2 = sigmoid(z2)$

$\mathrm{w}2 \in R^{10\times60}, b2 \in R^{10\times1}, z2 \in R^{10\times batch\ size},$
$a2\ \in R^{10\times batch\ size}$



$$sigmoid(x) = \frac{1}{1+e^{-x}}$$



$$relu(x) = \max(0, x)$$

损失函数使用交叉熵，其中y是标签

$$J(W, b\ ) = -yloga2$$

反向传播

$$x \rightarrow z1 = w1x + b1 \rightarrow a1 = relu(z1) \rightarrow z2 = w2a1 + b2 \rightarrow a2 = sigmoid(z2)$$

$$dw1 = \frac{1}{m} dz1x^T$$

$$dw2 = \frac{1}{m} dz2a1^T$$

$$dz1 = \begin{cases} da1 & if \;\; z1 > 0 \\ 0 & if \; z1 < 0 \end{cases}$$

$$da1 = w2^T dz2$$

$$dz2 = a2 - y$$

$$db1 = \frac{1}{m} np.sum(dz1, axis = 1)$$

$$db2 = \frac{1}{m} np.sum(dz2, axis = 1)$$

梯度更新

$$w1 := w1 - dw1$$
$$b1 := b1 - db1$$
$$w2 := w2 - dw2$$
$$b2 = b2 - db2$$

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):
[offset] [type] [value] [description]
0000 32 bit integer 0x00000803(2051) magic number
0004 32 bit integer 60000 number of images #图像个数
0008 32 bit integer 28 number of rows #图像宽度
0012 32 bit integer 28 number of columns #图像高度
0016 unsigned byte ?? pixel #图像像素值
0017 unsigned byte ?? pixel

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):
[offset] [type] [value] [description]
0000 32 bit integer 0x00000801(2049) magic number
0004 32 bit integer 60000 number of items
0008 unsigned byte ?? label
0009 unsigned byte ?? label
....
xxxx unsigned byte ?? label

```python
import struct
import numpy as np
```

```python
learn_rate = 0.001
```

前两行数据都是32位，所以使用两个I

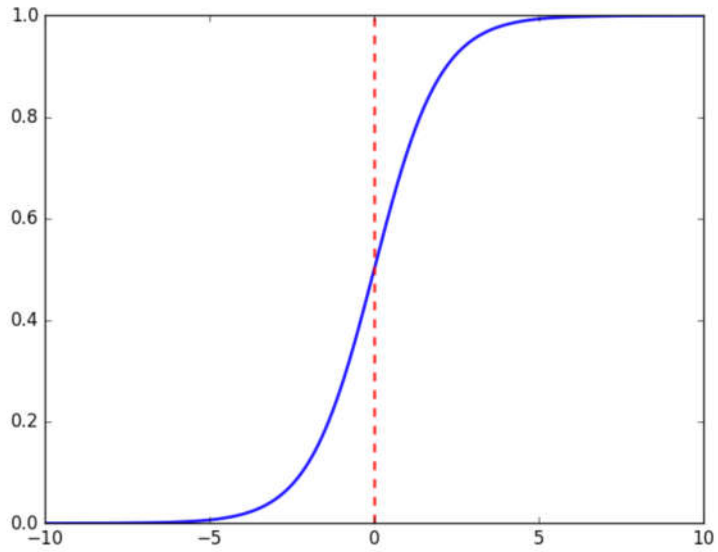
```python
def get_data():
    with open('train-labels.idx1-ubyte', 'rb') as lbpath:
        magic, n = struct.unpack('>II', lbpath.read(8))
        labels = np.fromfile(lbpath, dtype=np.uint8)

    with open('train-images.idx3-ubyte', 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack('>IIII', imgpath.read(16))
        images = np.fromfile(imgpath, dtype=np.uint8).reshape(len(labels), 784)

    list_labels = np.zeros([60000, 10])
    for index in range(60000):
        list_labels[index][labels[index]] = 1
    return list_labels, images
```

一次从文件中读取8个字节

```python
def parameter_initialization():
    w1 = 0.001*np.random.rand(60, 784)
    w2 = 0.001*np.random.rand(10, 60)
    b1 = 0.001*np.random.randn(60, 1)
    b2 = 0.001*np.random.randn(10, 1)
    return w1, w2, b1, b2
```




```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

1.初始化为0，所有的隐藏单元都是对称的，无论运行多久，他们计算的都是一样的函数

2. 初始数值比较大的时候，在使用sigmoid 函数，会停留在比较平坦的地方，梯度下降会很慢

```python
def relu(x):
    return np.maximum(0,x)
```

```python
def relu_backward(next_dz,z):
    return np.where(np.greater(z, 0), next_dz, 0)
```

```python
def buildmode(images, labels, w1, w2, b1, b2):
    images = images.T
    labels = labels.T
    batch_size = 250
    for batch in range(int(images.shape[-1]/batch_size)):
        start = batch*batch_size
        batchImage = images[:,start:start+batch_size]
        batchlabel = labels[:,start:start+batch_size]
        z1 = np.dot(w1,batchImage) + b1
        a1 = relu(z1)  # 输入层输出
        z2 = np.dot(w2, a1)+b2
        a2 = sigmoid(z2)  # 隐二层输出w
        loss = -batchlabel*np.log(a2)
        dz2 = a2 - batchlabel
        dw2 = np.dot(dz2,a1.T)/batch_size
        db2 = np.sum(dz2,axis=1,keepdims=True)/batch_size
        da1 = np.dot(w2.T,dz2)
        dz1 = relu_backward(da1,z1)
        dw1 = np.dot(dz1,batchImage.T)/batch_size
        db1 = np.sum(dz1,axis=1,keepdims=True)/batch_size
        w1 = w1 - learn_rate * dw1
        w2 = w2 - learn_rate * dw2
        b1 = b1 - learn_rate * db1
        b2 = b2 - learn_rate * db2
        if batch % 10 == 0:
            print('第{a}batch训练的当前的loss值为{b}'.format(a=batch, b=np.sum(loss)/batch_size))
    return w1, w2,  b1, b2
```

正向传播

反向传播

参数更新