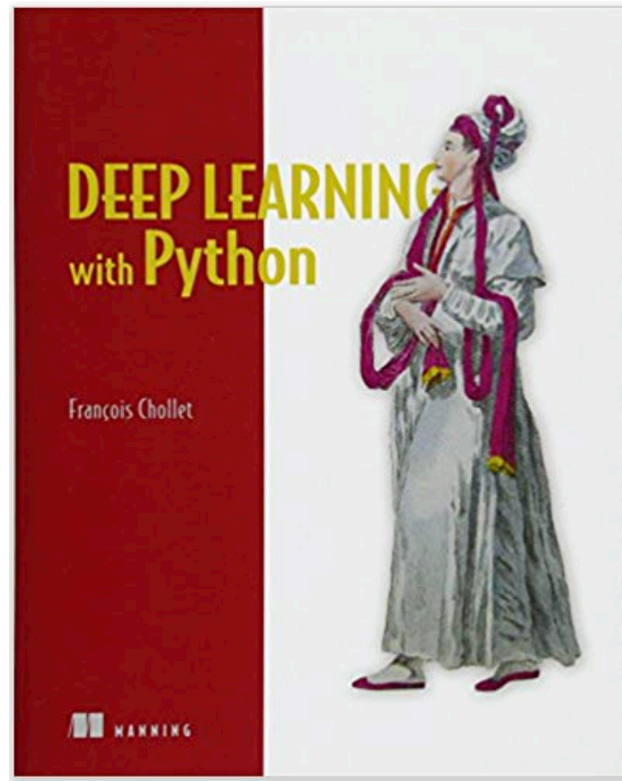
CSCE 636 Neural Networks (Deep Learning)

Lecture 2: Mathematical Building Blocks of Neural Networks

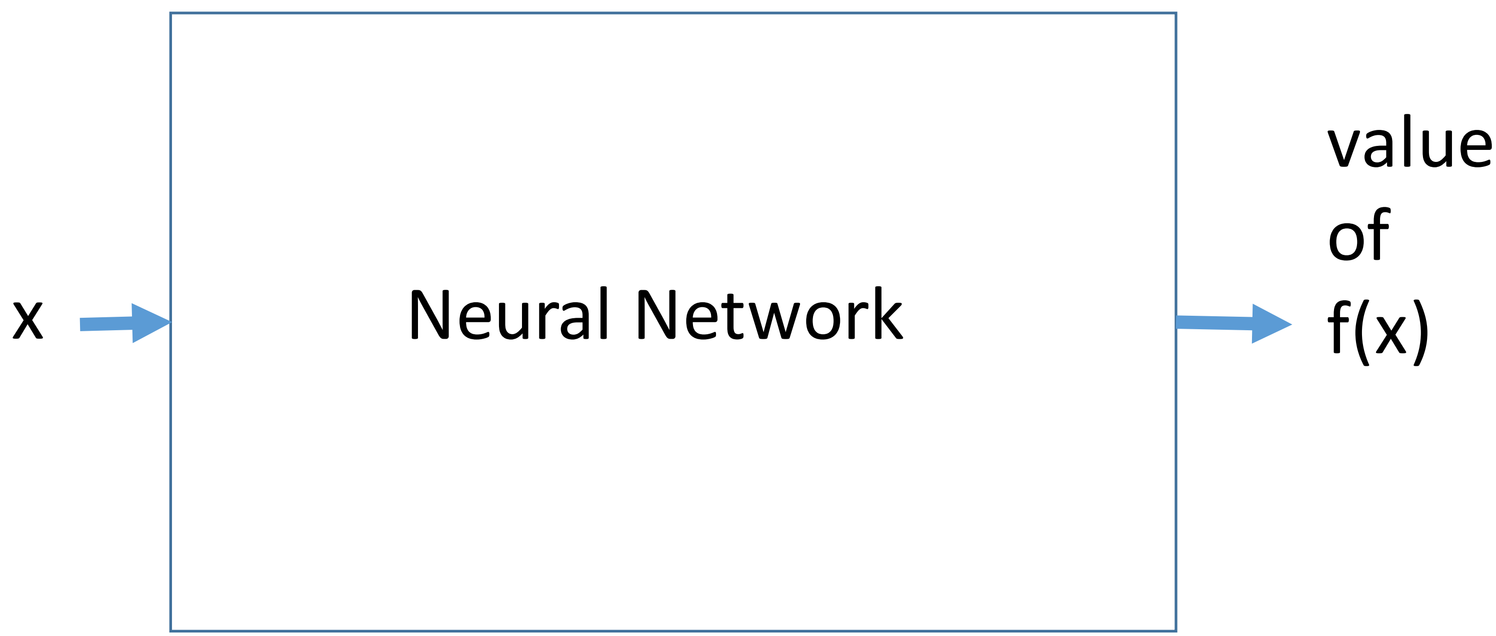
Anxiao (Andrew) Jiang

Chapter 2

Before we begin: the mathematical building blocks of neural networks



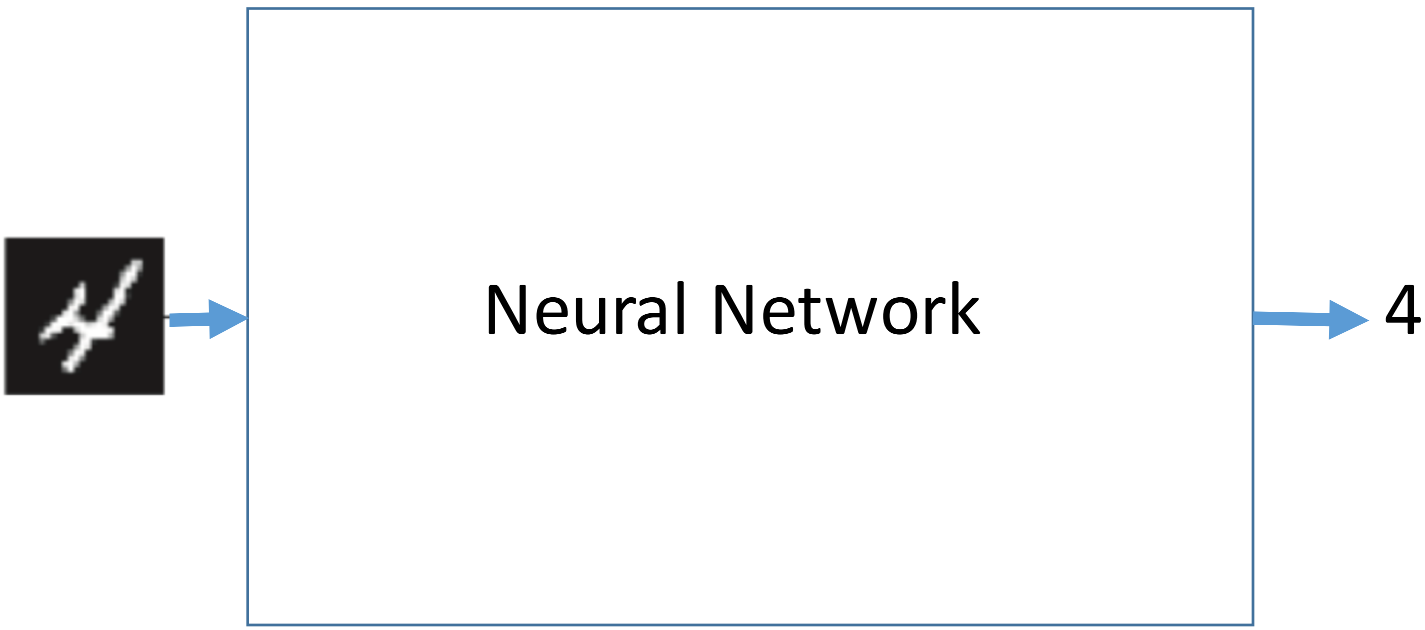
What a neural network does: learn a function



The neural network learns the function f(x), either exactly or approximately.

Application: Handwritten Digit Recognition

Task: Classify grayscale images of handwritten digits (28x28 pixels) into their 10 categories (0 through 9).



How to start?

Step 1: Load the dataset

MNIST Dataset: 60,000 training images and 10,000 test images, along with their labels.

Step 1: Load the dataset

Listing 2.1 Loading the MNIST dataset in Keras

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

train_images: 60,000 x 28 x 28 array, where each element (pixel) is an integer in [0,255]
train_labels: vector of length 60,000, where each element (label) is an integer in [0,9]

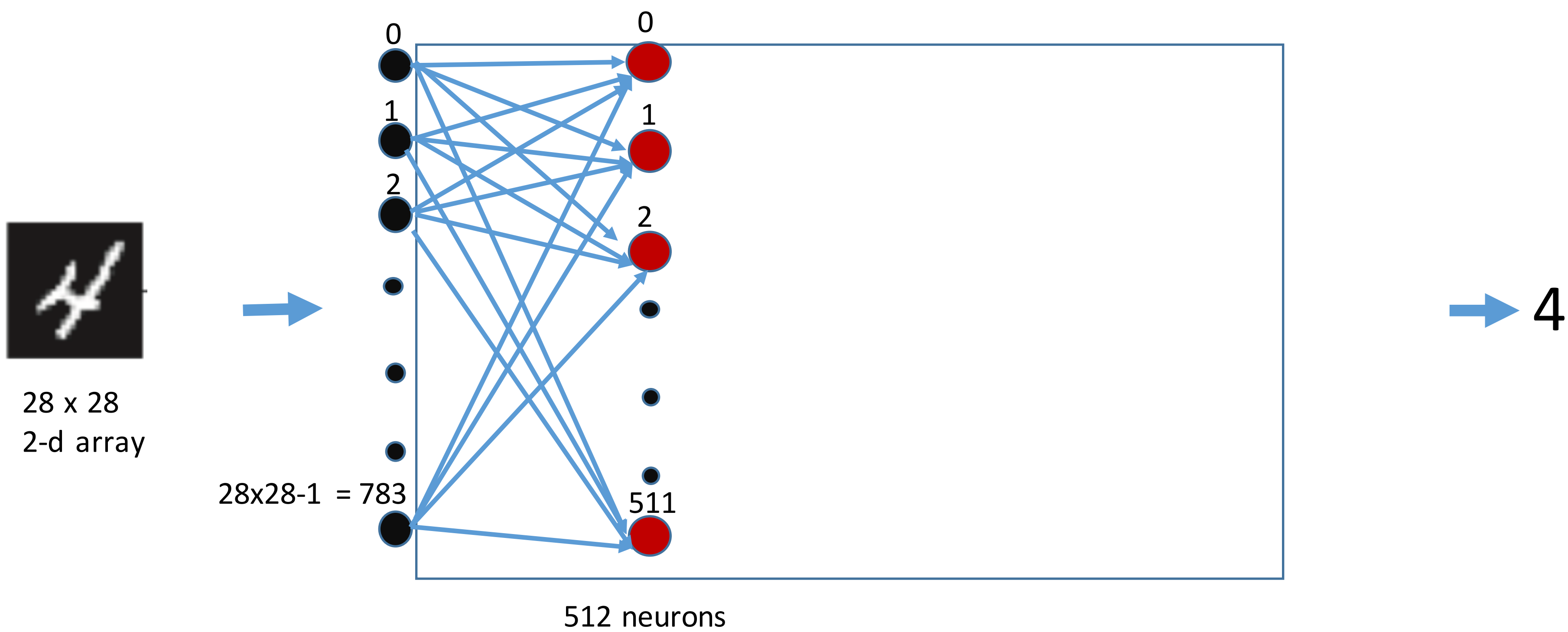
test_images: 10,000 x 28 x 28 array, where each element (pixel) is an integer in [0,255]
test_labels: vector of length 10,000, where each element (label) is an integer in [0,9]

Rule: training data and test data are disjoint. Only use training data to train neural network!

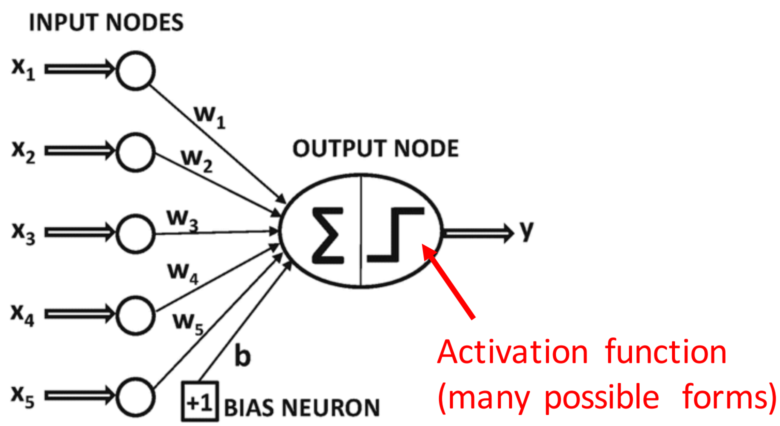

4

Step 2: Build neural network architecture

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
```



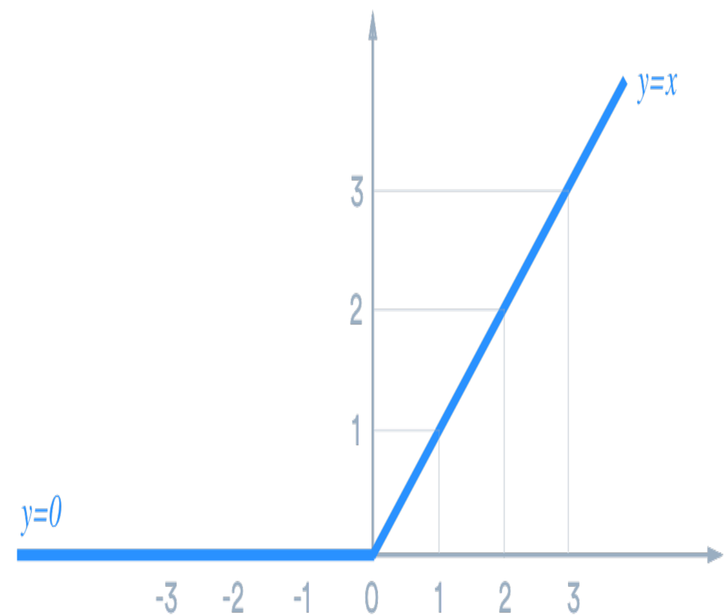
What is a neuron



(b) Perceptron with bias

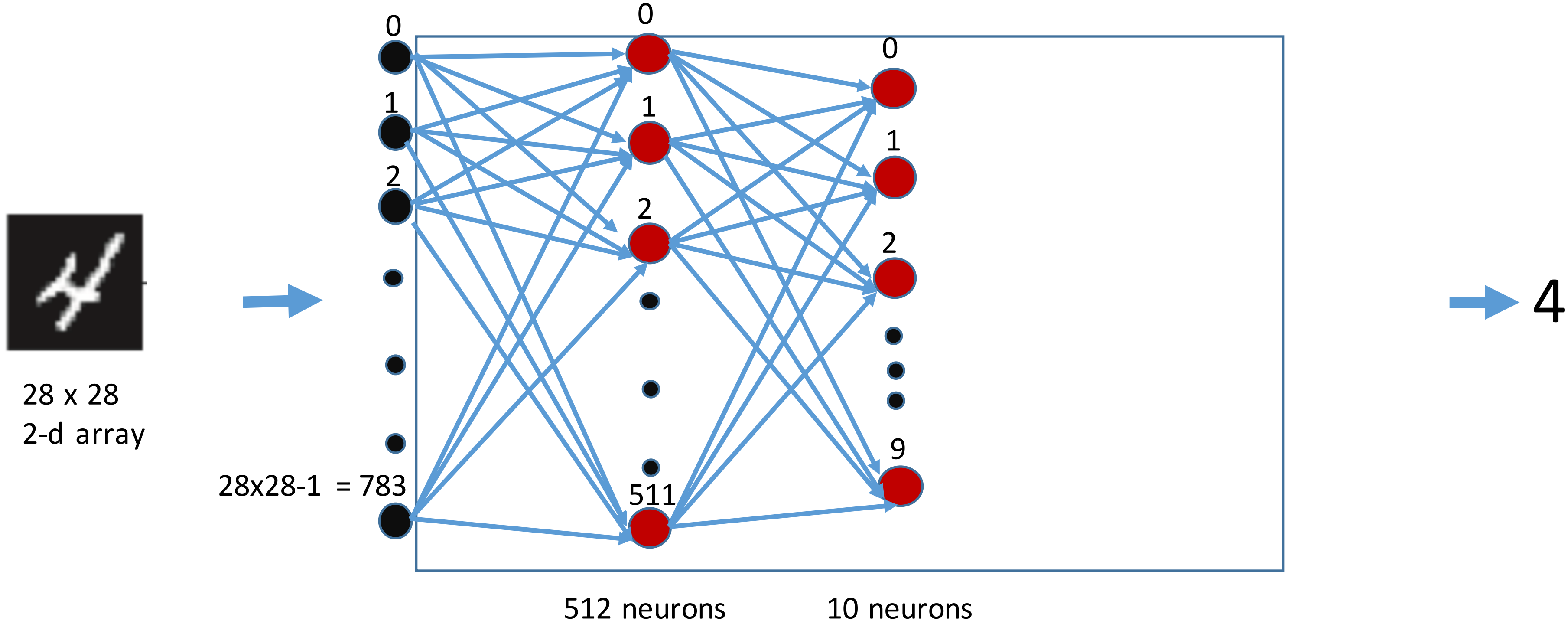
$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j + b\}$$

ReLU (most popular Activation function)

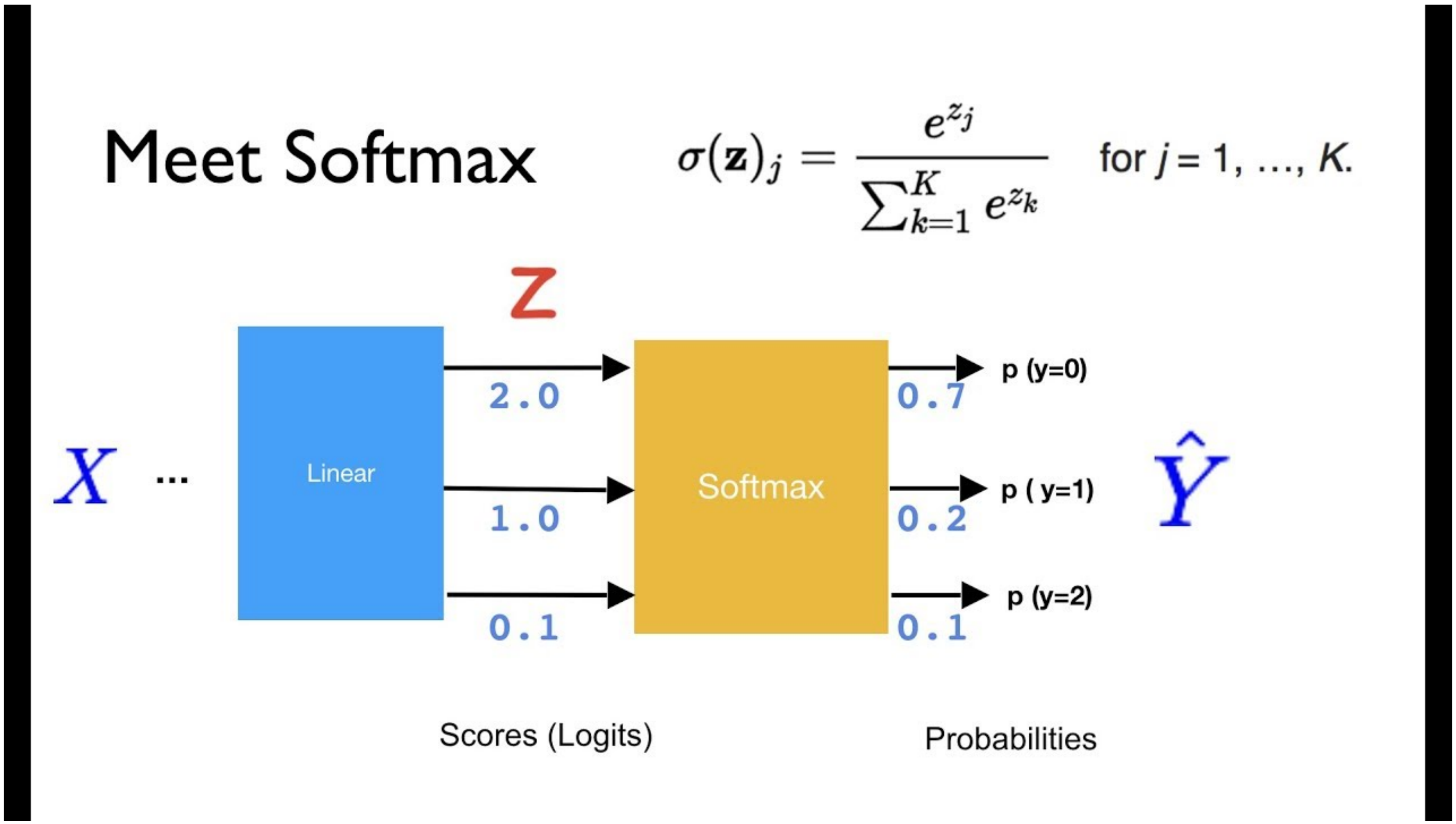


Step 2: Build neural network architecture

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

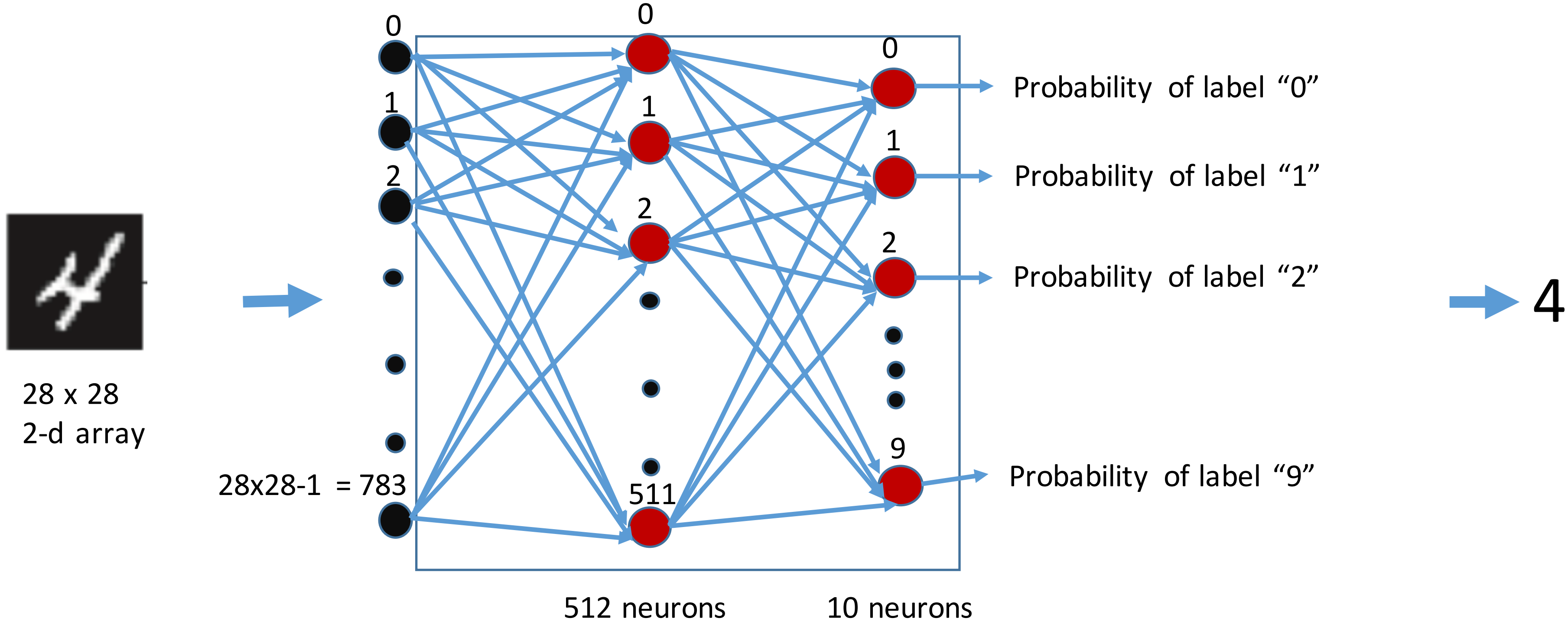


Softmax (popular activation function for the last layer of a classification network)



Step 2: Build neural network architecture

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```



Step 3: choose loss function, optimizer, and target metrics

Listing 2.3 The compilation step

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Categorical cross-entropy (a popular loss function for multi-class classification)

Number of classes

$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)$$

probability predicted by neural network that this input belongs to class j

Number of samples

True probability (1 or 0) this input belongs to class j

RMSProp (a popular optimizer, details to be introduced later)

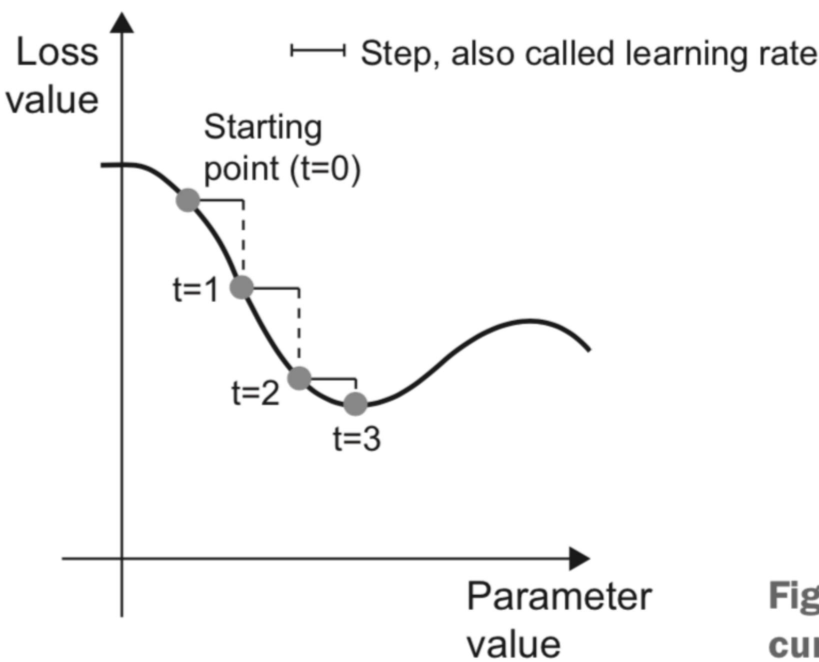
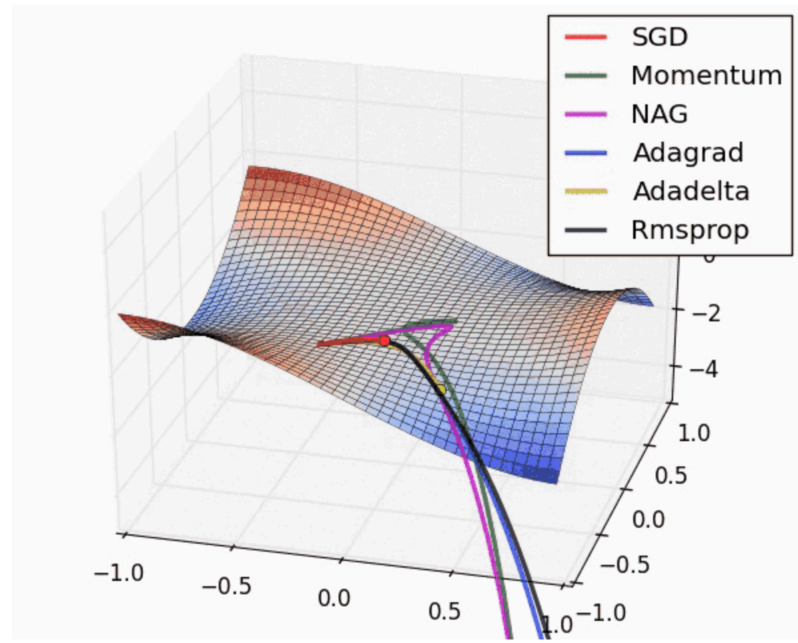


Figure 2.11 SGD down a 1D loss curve (one learnable parameter)

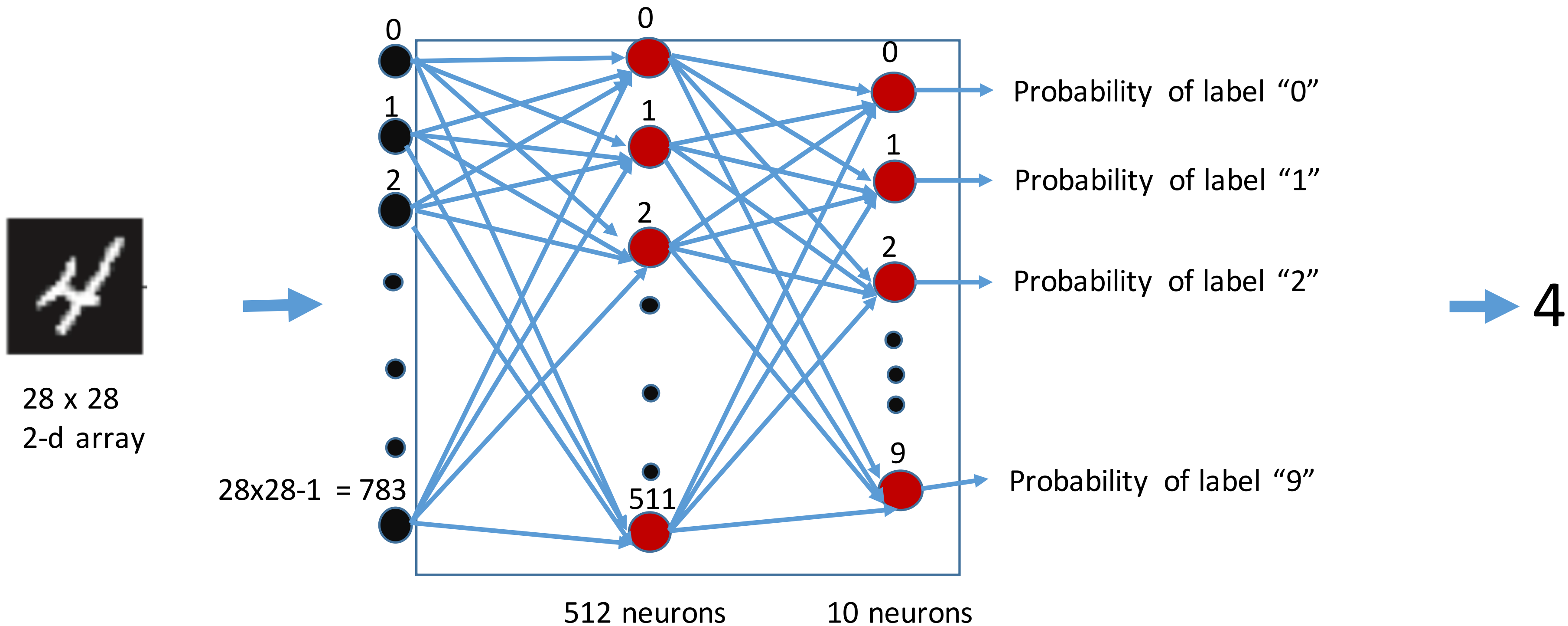


Accuracy: fraction of times that the neural network makes correction predictions

- If we care about accuracy, why do we optimize categorical cross-entropy during training?
- Answer: loss function needs to be differentiable. (And the loss function is closely related to the target metric. Minimizing the loss function is (approximately or precisely) optimizing the target metric.

The “Teacher”:

Loss function: categorical cross-entropy
Optimizer: RMSProp
Target Metric: Accuracy



Step 4: Prepare training and test data

Here: Reshape and normalize input training data

Listing 2.4 Preparing the image data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

train_images:
Originally: 3-dimensional array of size 60000 x 28 x 28, where each element is an integer in [0,255]
After reshaping: 2-dimensional array of size 60000 x 784, where each element is an integer in [0,255]
After normalization: 2-dimensional array of size 60000 x 784, where each element is a real number in [0,1]

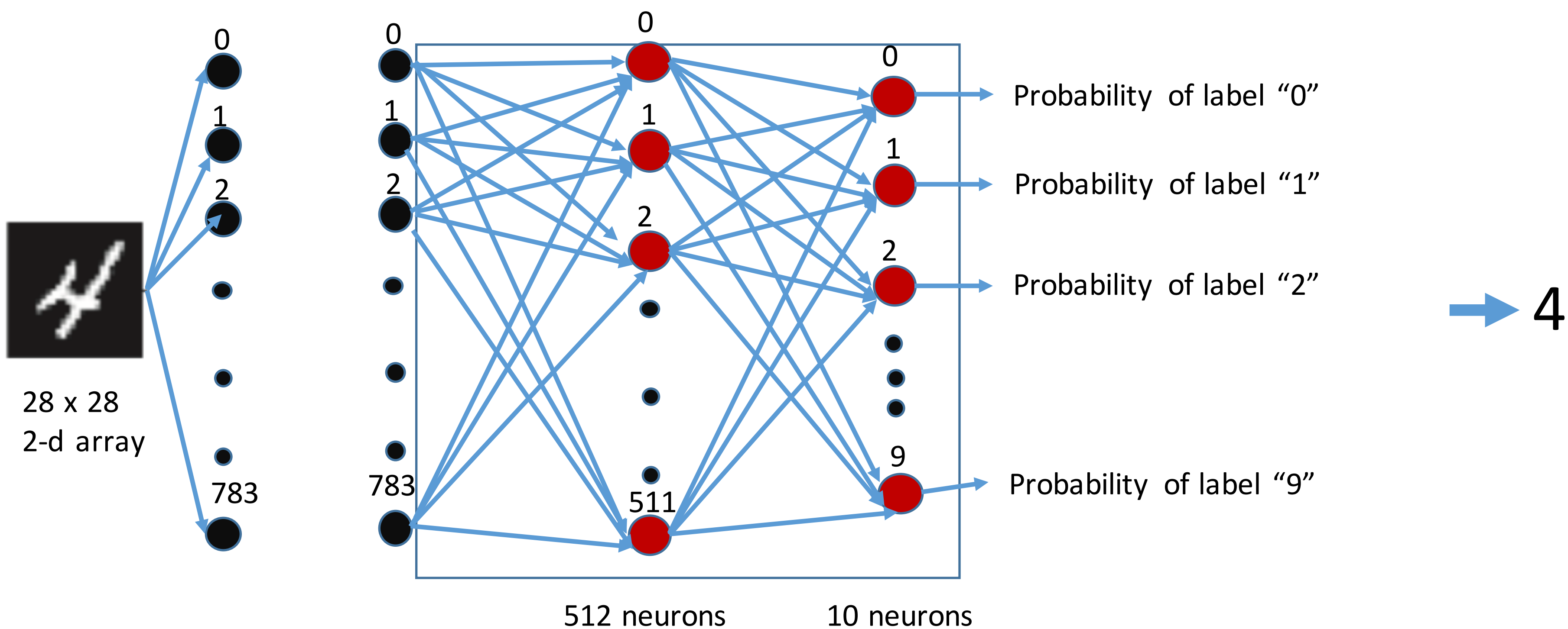
28 x 28
2-d array

→ 1-d array of length
28*28 = 784

→ Normalize values in the array to between 0 and 1

The “Teacher”:

Loss function: categorical cross-entropy
Optimizer: RMSProp
Target Metric: Accuracy



“Reshape” output training data: categorically encode each label using one-hot encoding

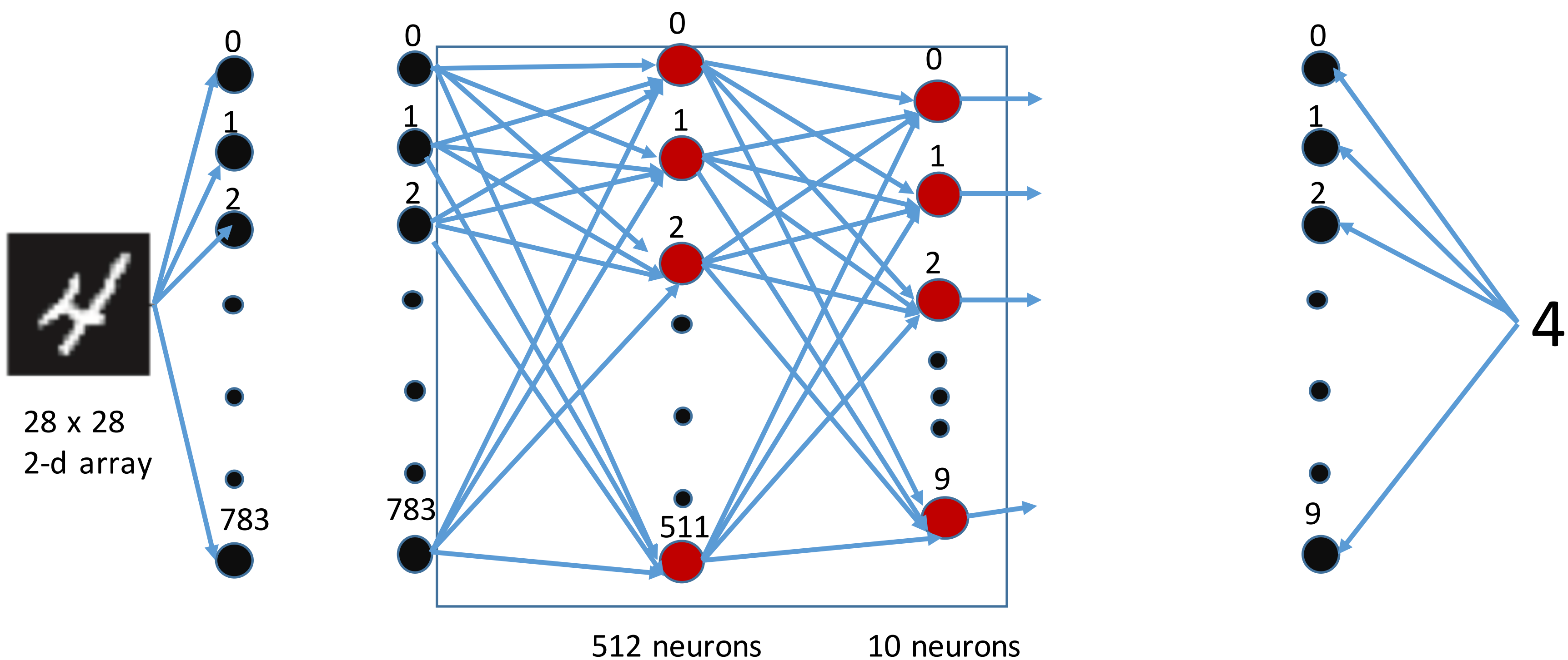
```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Label		One-hot encoding
0	→	1,0,0,0,0,0,0,0,0,0
1	→	0,1,0,0,0,0,0,0,0,0
2	→	0,0,1,0,0,0,0,0,0,0
3	→	0,0,0,1,0,0,0,0,0,0
4	→	0,0,0,0,1,0,0,0,0,0

Label		One-hot encoding
5	→	0,0,0,0,0,1,0,0,0,0
6	→	0,0,0,0,0,0,1,0,0,0
7	→	0,0,0,0,0,0,0,1,0,0
8	→	0,0,0,0,0,0,0,0,1,0
9	→	0,0,0,0,0,0,0,0,0,1

The “Teacher”:

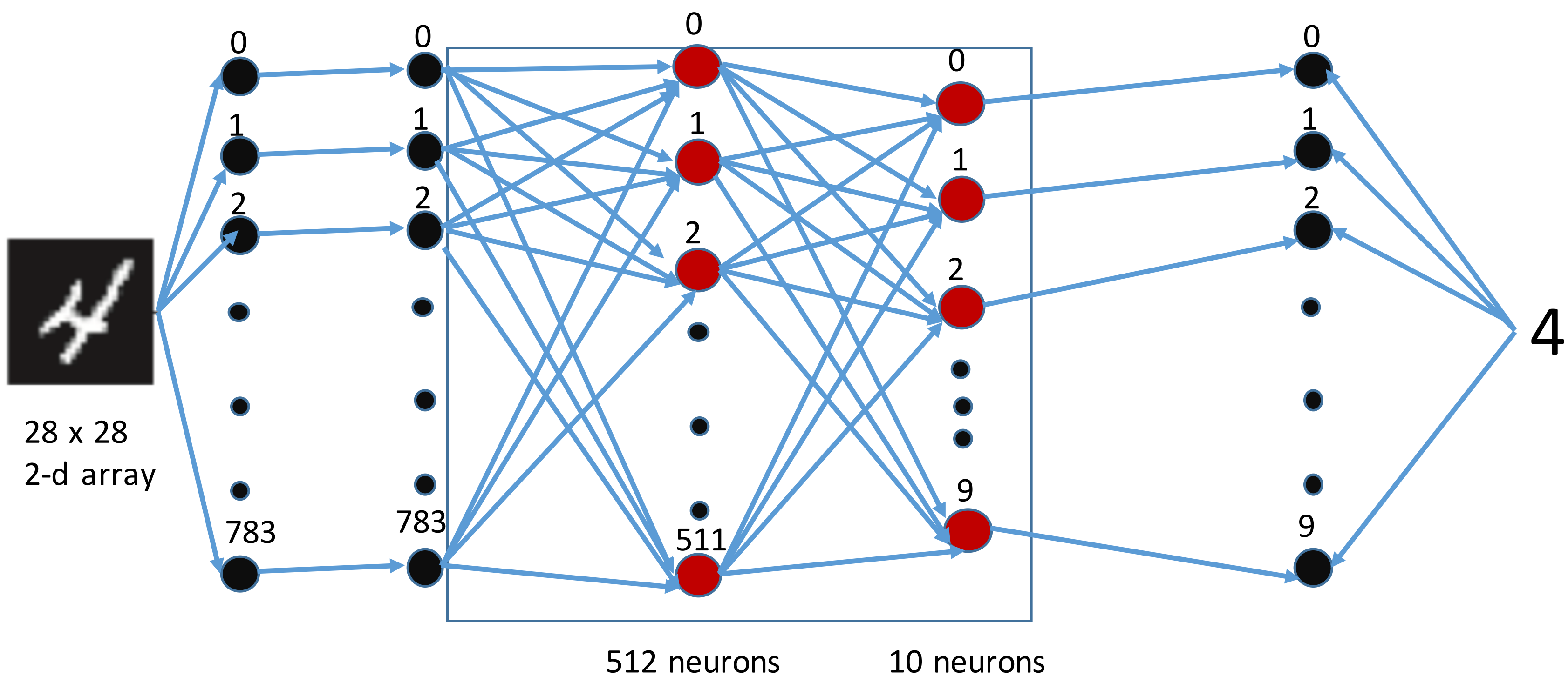
Loss function: categorical cross-entropy
Optimizer: RMSProp
Target Metric: Accuracy



Step 5: Train the neural network

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

The "Teacher": Loss function, Optimizer, Target Metric: Accuracy



Batch size: the number of samples to use each time for computing the loss function and updating the weights.

Epochs: the number of times the training process uses the whole training data set.

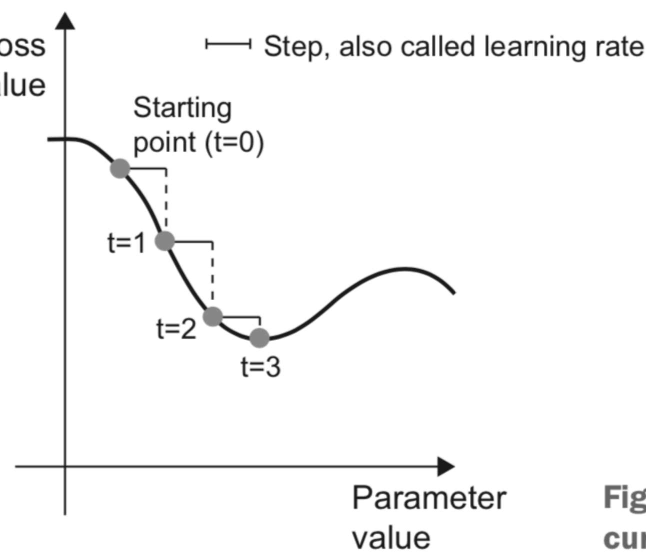


Figure 2.11 SGD down a 1D loss curve (one learnable parameter)

```
>>> network.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [==============================] - 9s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [========================>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

And so on (totally 5 epochs).

Accuracy on training data: 97.8%

Step 6: Test the trained neural network

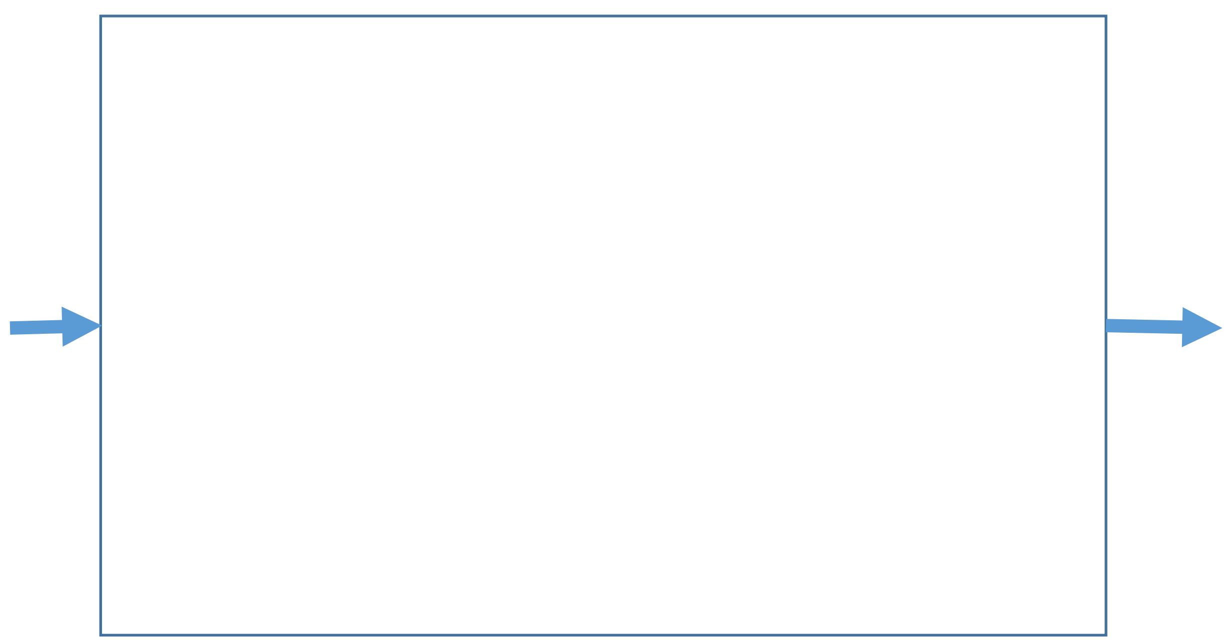
```
>>> test_loss, test_acc = network.evaluate(test_images, test_labels)
>>> print('test_acc:', test_acc)
test_acc: 0.9785
```

Compare to training accuracy: 0.989

Test accuracy is (clearly) lower than training accuracy.
Maybe there is some over-fitting to data.

But still, performance is nice!

Summary



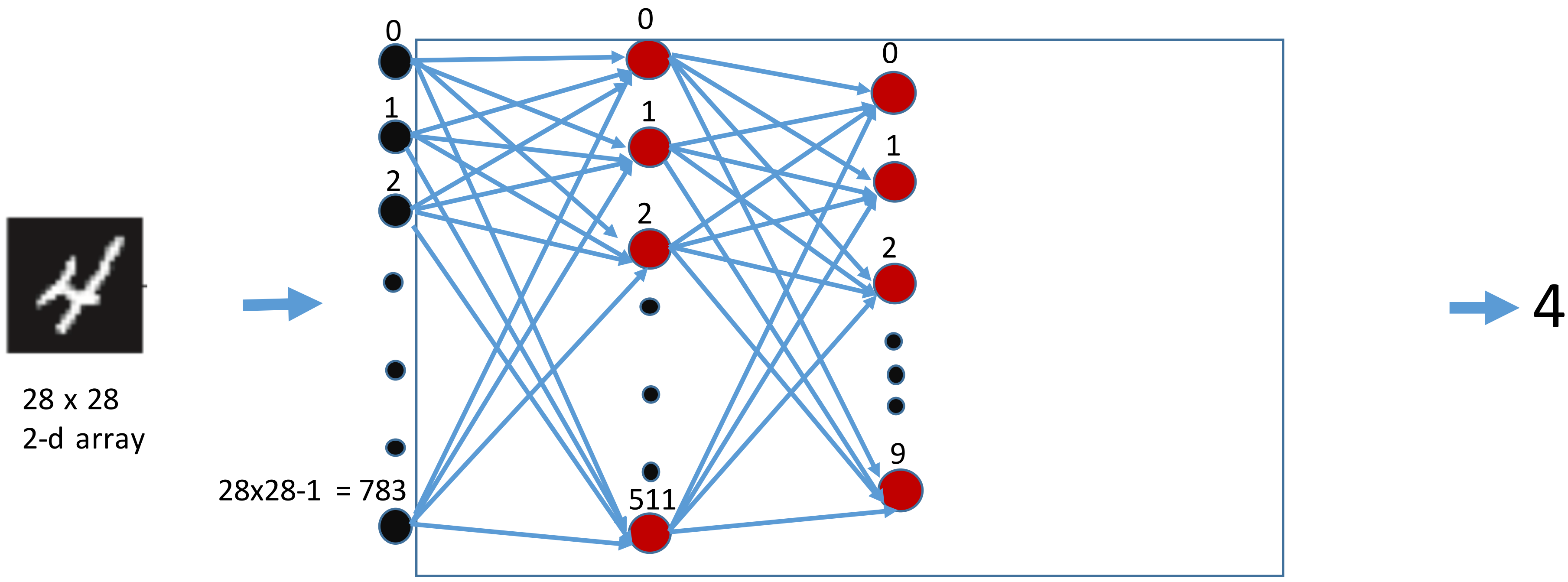
Step 1: Load the dataset

Listing 2.1 Loading the MNIST dataset in Keras

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Step 2: Build neural network architecture

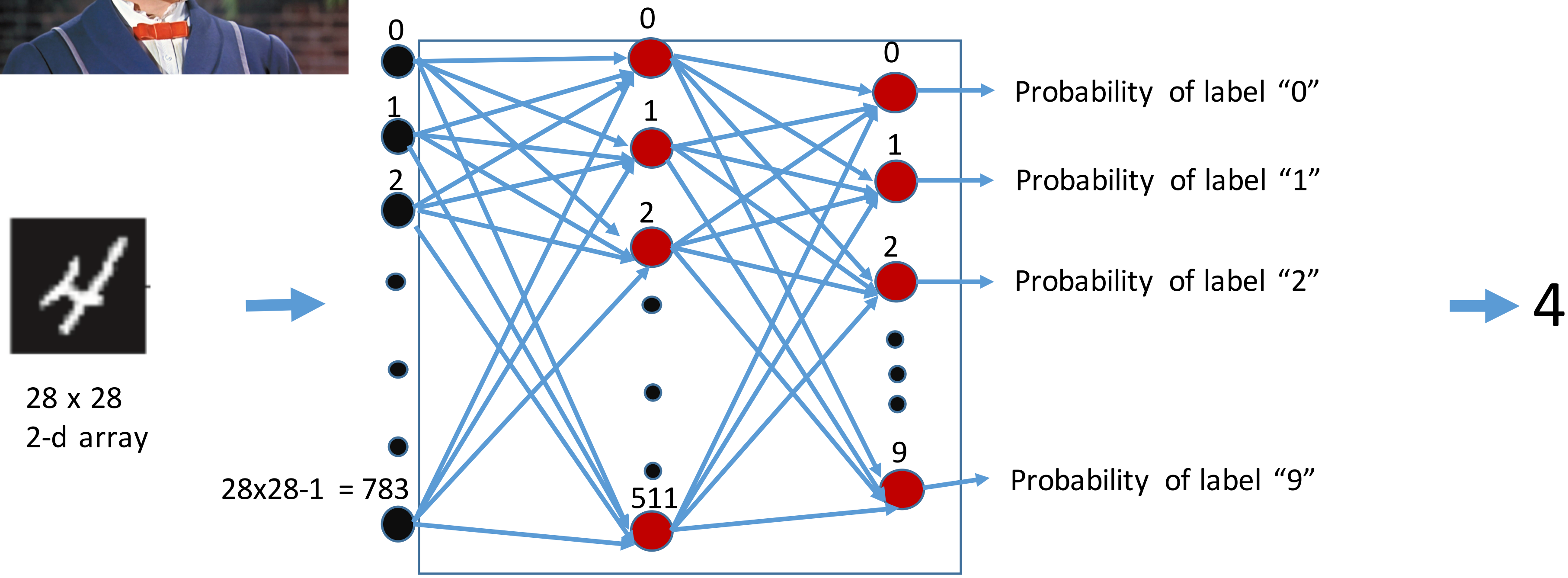
```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```



Step 3: choose loss function, optimizer, and target metrics

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

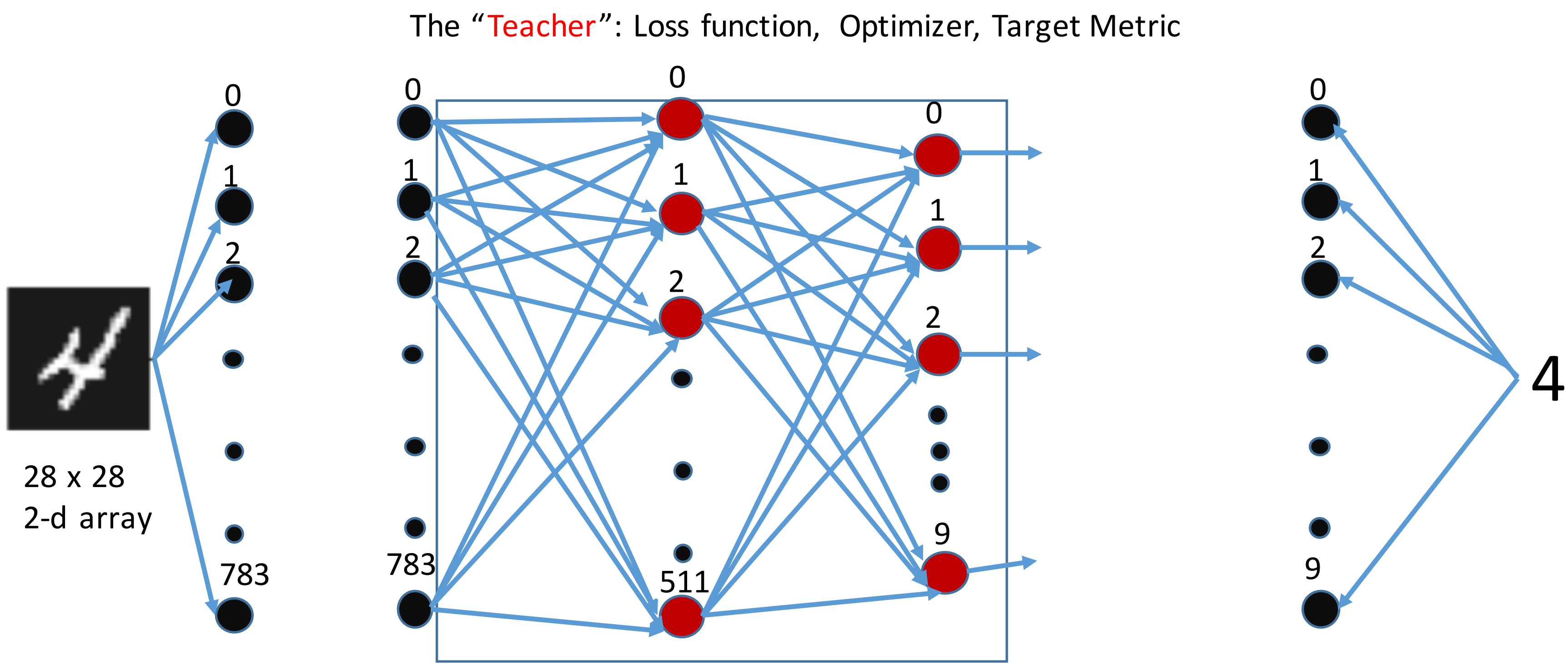
The “Teacher”: Loss function, Optimizer, Target Metric



Step 4: Prepare training and test data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

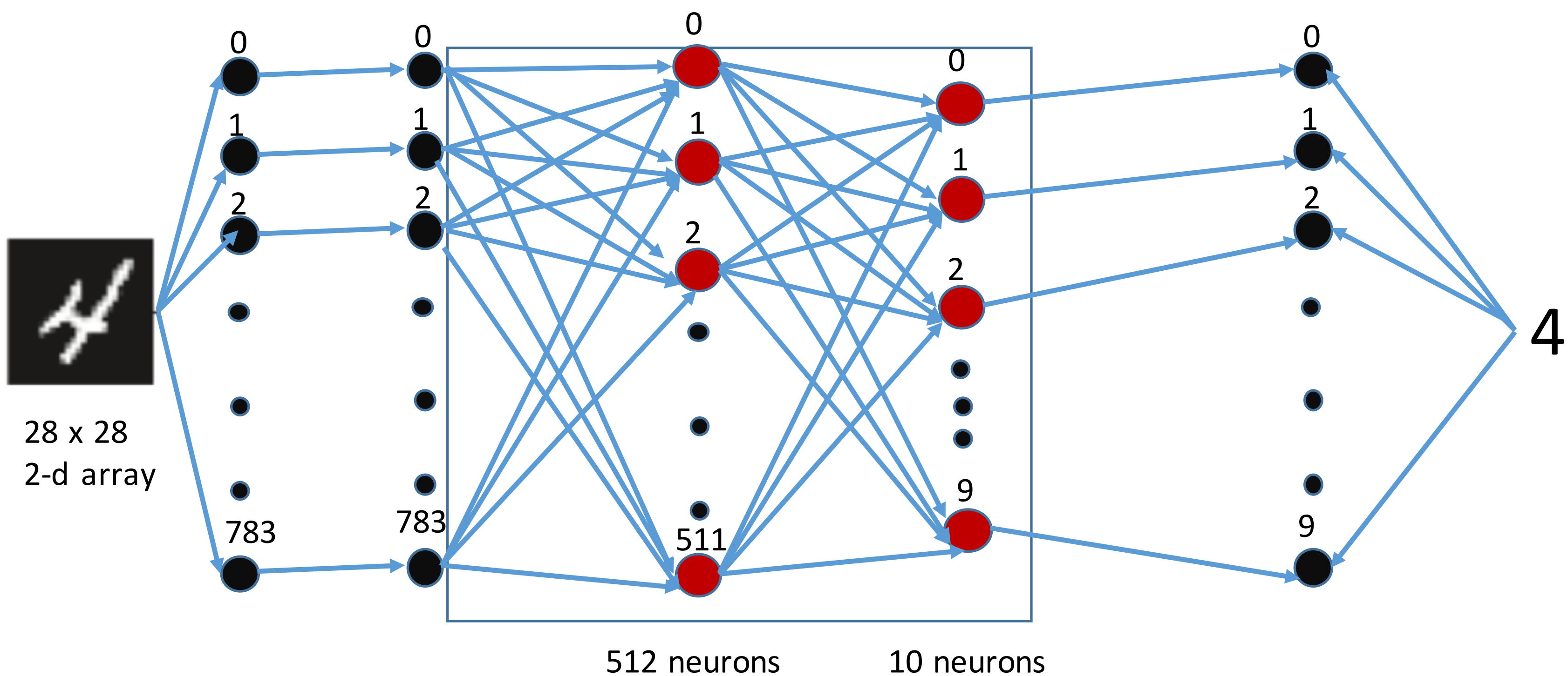
```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```



Step 5: Train the neural network

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

The “Teacher”: Loss function, Optimizer, Target Metric: Accuracy



Step 6: Test the trained neural network

```
>>> test_loss, test_acc = network.evaluate(test_images, test_labels)
>>> print('test_acc:', test_acc)
test_acc: 0.9785
```

How did I do?

Well …

Miscellaneous Basic Concepts

Data representation: Tensor (Array)

- Scalar numbers (0-dimensional tensors)
- Vectors (1-d tensors)
- Matrices (2-d tensors)
- 3-d tensors, and higher-dimensional tensors

- Key attributes for a tensor:
- (1) number of axes
- (2) shape
- (3) data type

Some basic tensor operations

- Add two tesnors (of the same shape): element-wise addition
- Apply a ReLU activation function to a tensor: element-wise operation

- Tensor Product (also called tensor dot)

$$A_5^T B_5 = C \text{ (scalar number)}$$

$$A_{7,5} B_5 = C_7$$

$$A_7^T B_{7,5} = C_5^T$$

$$A_{2,8} B_{8,6} = C_{2,6}$$

$$A_{2,3,4,5} B_5 = C_{2,3,4}$$

$$A_{2,3,4,5} B_{5,6} = C_{2,3,4,6}$$

$$A_{2,3,4,5} B_{5,6,7} = C_{2,3,4,6,7}$$

$$A_{2,3,4,5} B_{5,4,7} = C_{2,3,4,4,7}$$

Reshape tensor

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)
```

```
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
```

```
>>> x = x.reshape((2, 3))
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
```

Basic terms for a neural network

- Layers: the building blocks in a neural network
- Model: network of layers
- Loss function and optimizer: keys to configuring the learning process

Keras: a deep learning library for Python

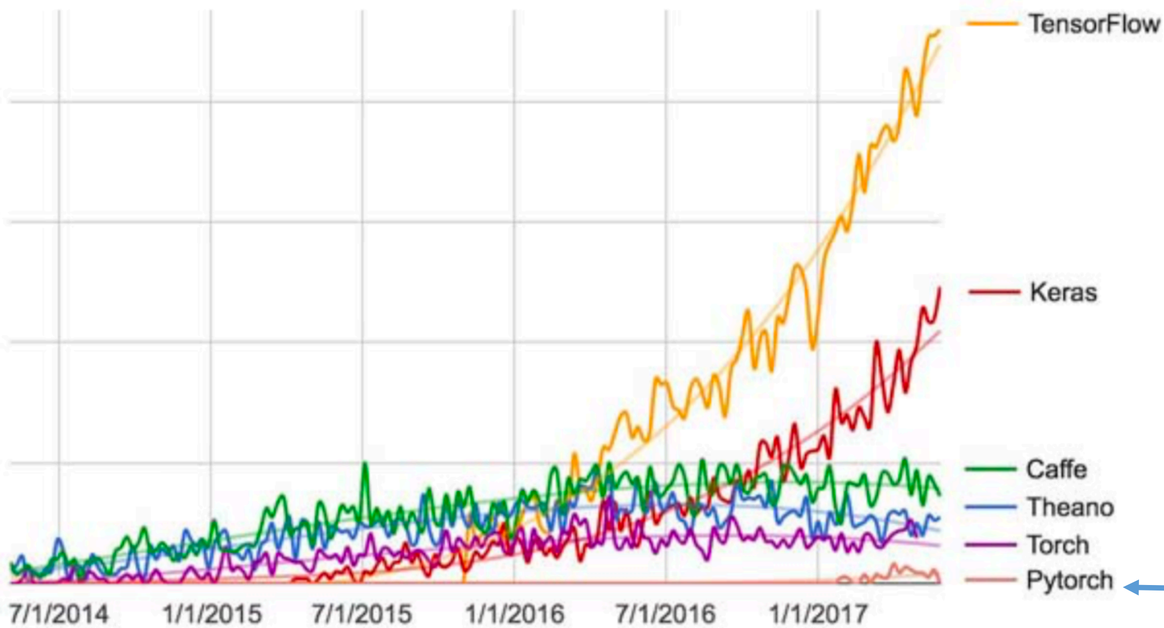


Figure 3.2 Google web search interest for different deep-learning frameworks over time

PyTorch is getting popular today

Keras: a deep learning library for Python

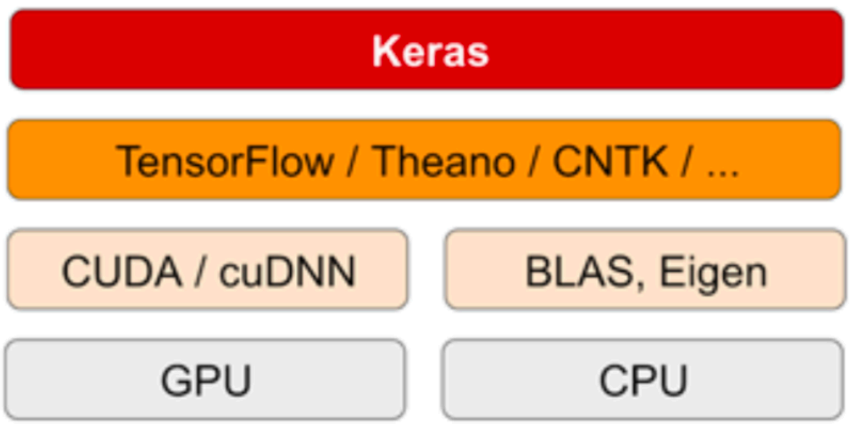


Figure 3.3 The deep-learning software and hardware stack

Use a GPU when possible

Jupyter notebook:
a nice way to edit and run deep learning experiments