Deep Learning

Lecture Topic:
Mathematical Building Blocks of Neural Networks

Anxiao (Andrew) Jiang

Learning Objectives:

1. Learn to build and train a basic neural network.

2. Understand basic concepts in deep learning.

Roadmap of this lecture:

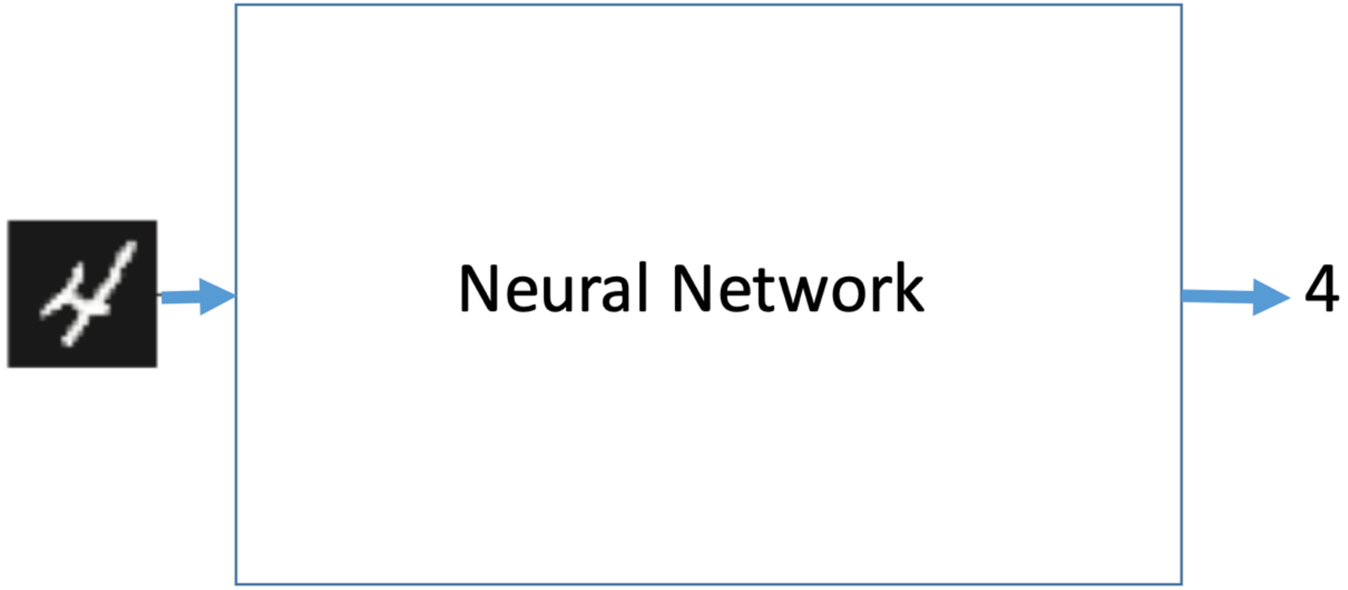
1. Handwritten digit recognition.

2. Basic concepts in deep learning.

Task:

Handwritten Digit Recognition

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



Step 1: Load the dataset

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

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.

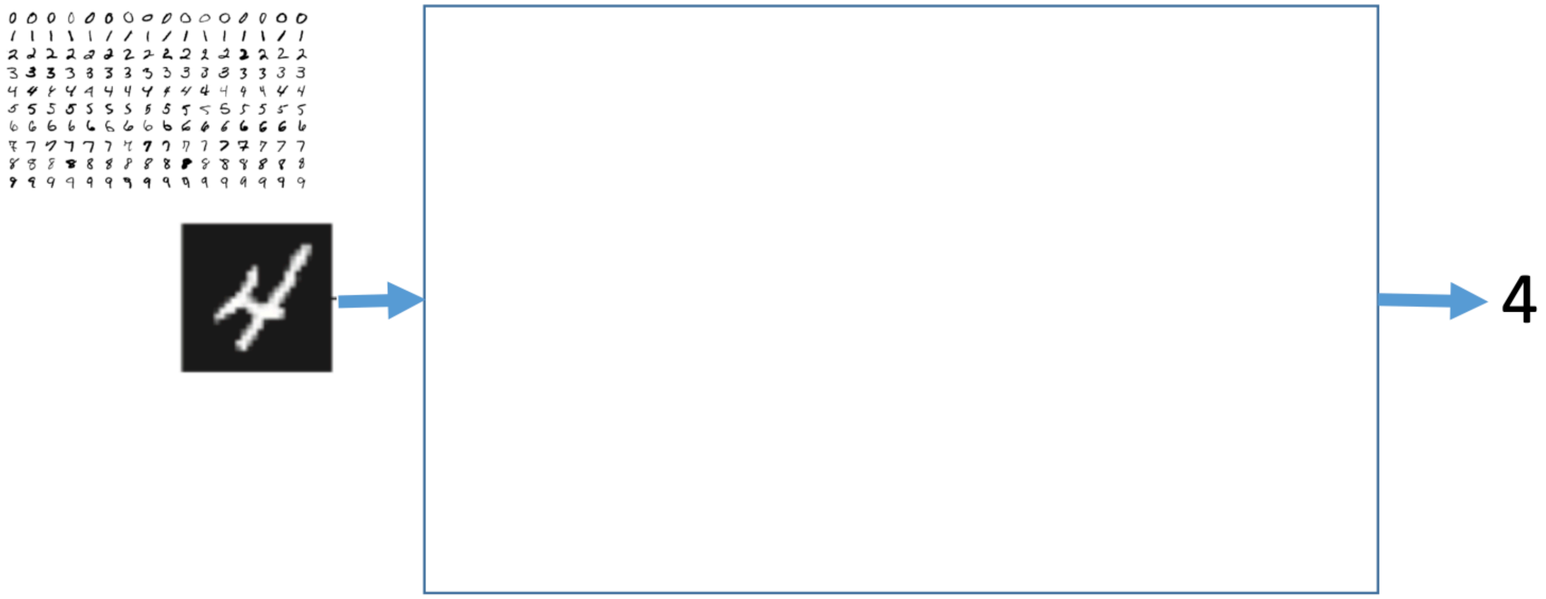
Each element (pixel) is an integer in [0, 255].

train_labels: a vector of length 60,000.

Each element (label) is an integer in [0, 9].

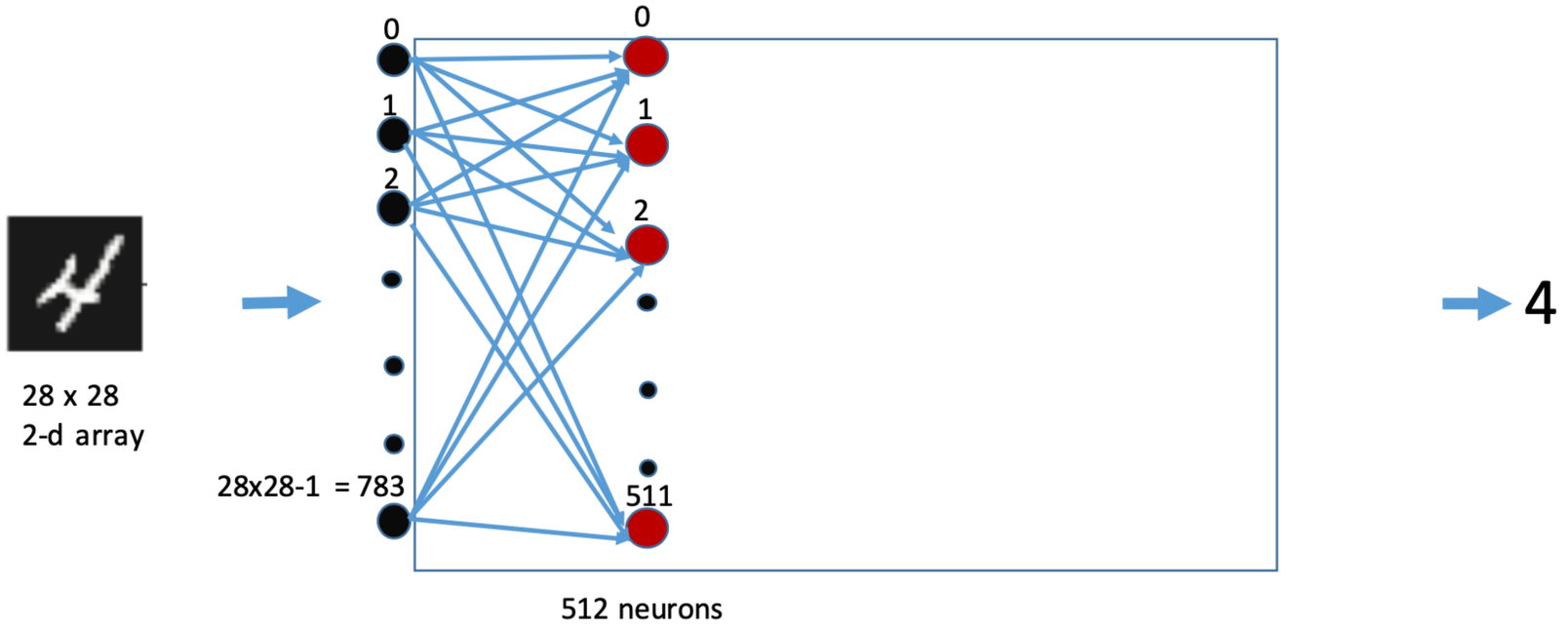
Testing set: 10,000 samples.

Training set and testing set are disjoint!

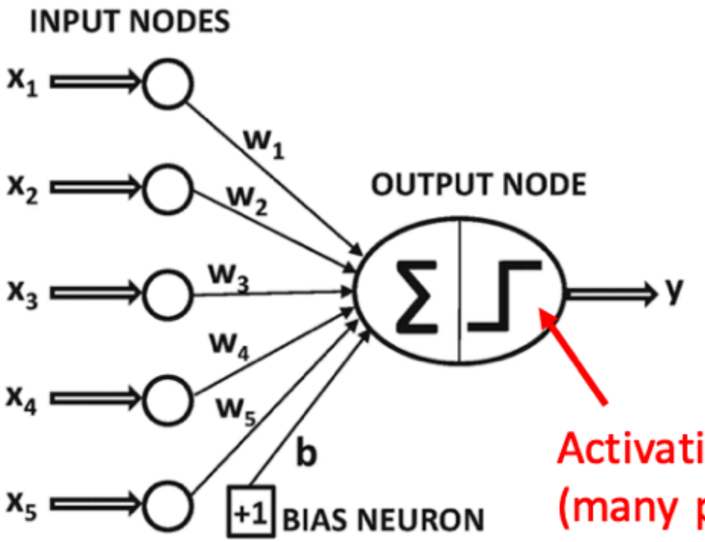

4

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,)))
```



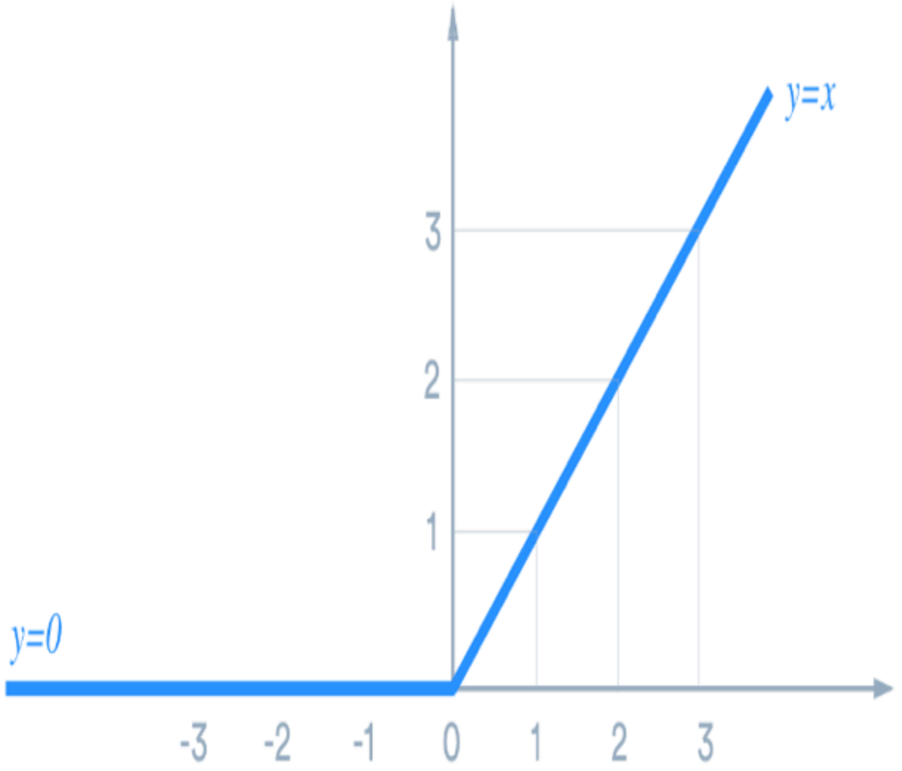
What is a neuron



Perceptron with bias

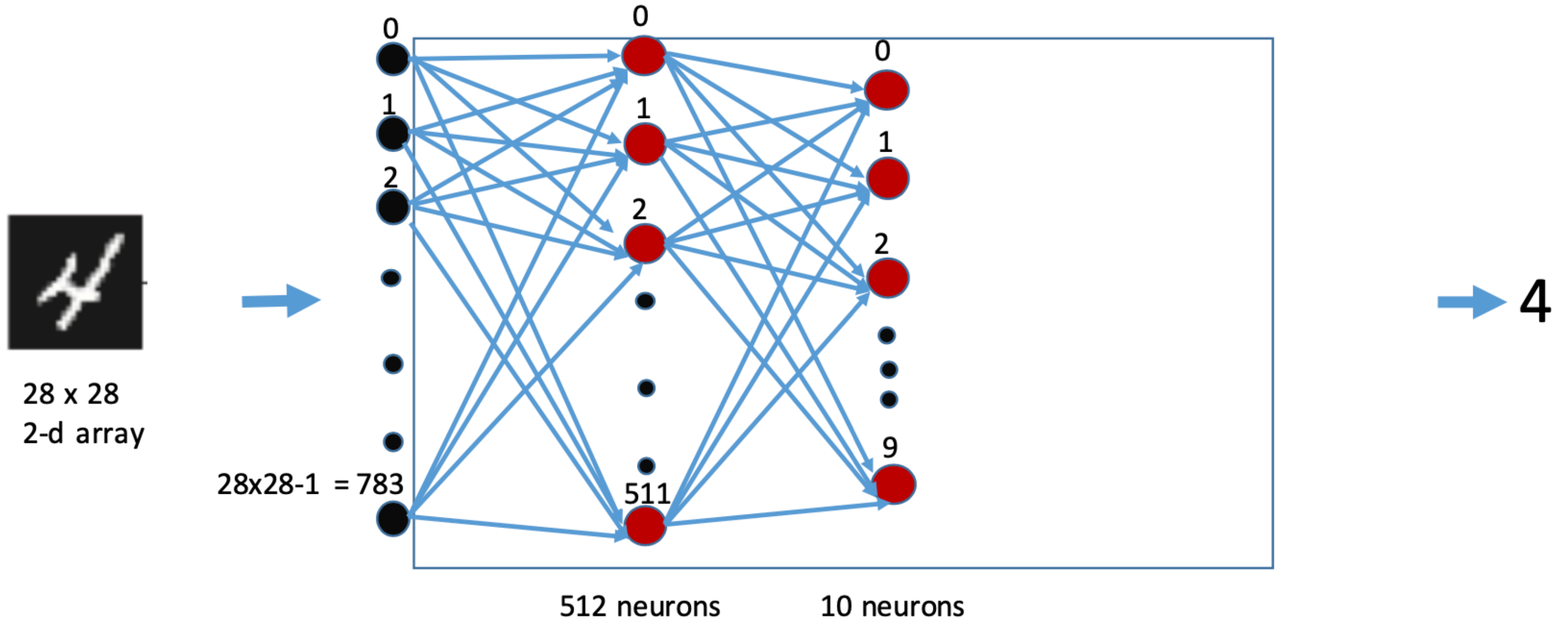
$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j + b\}$$

ReLU (most popular Activation function)

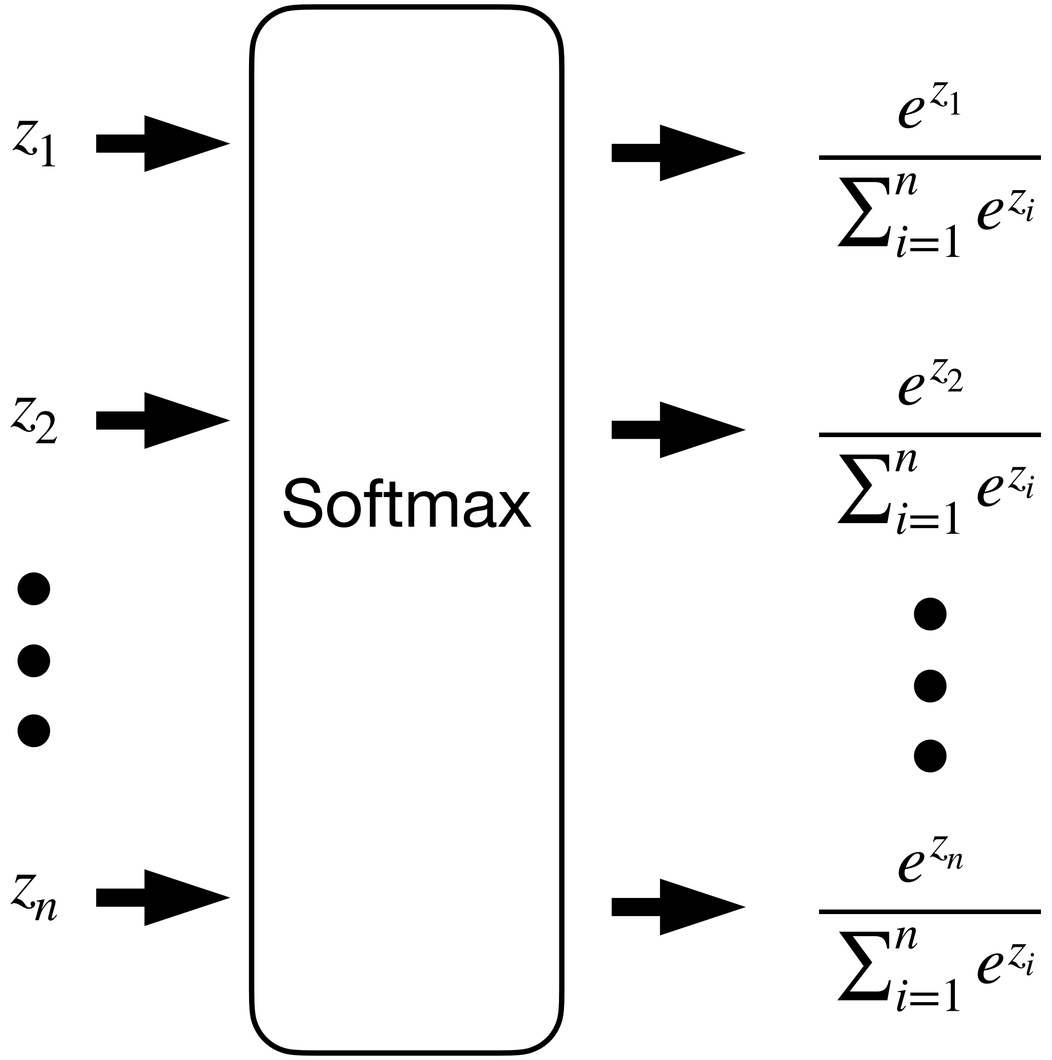


Step 2: Build neural network architecture

```
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'))
```

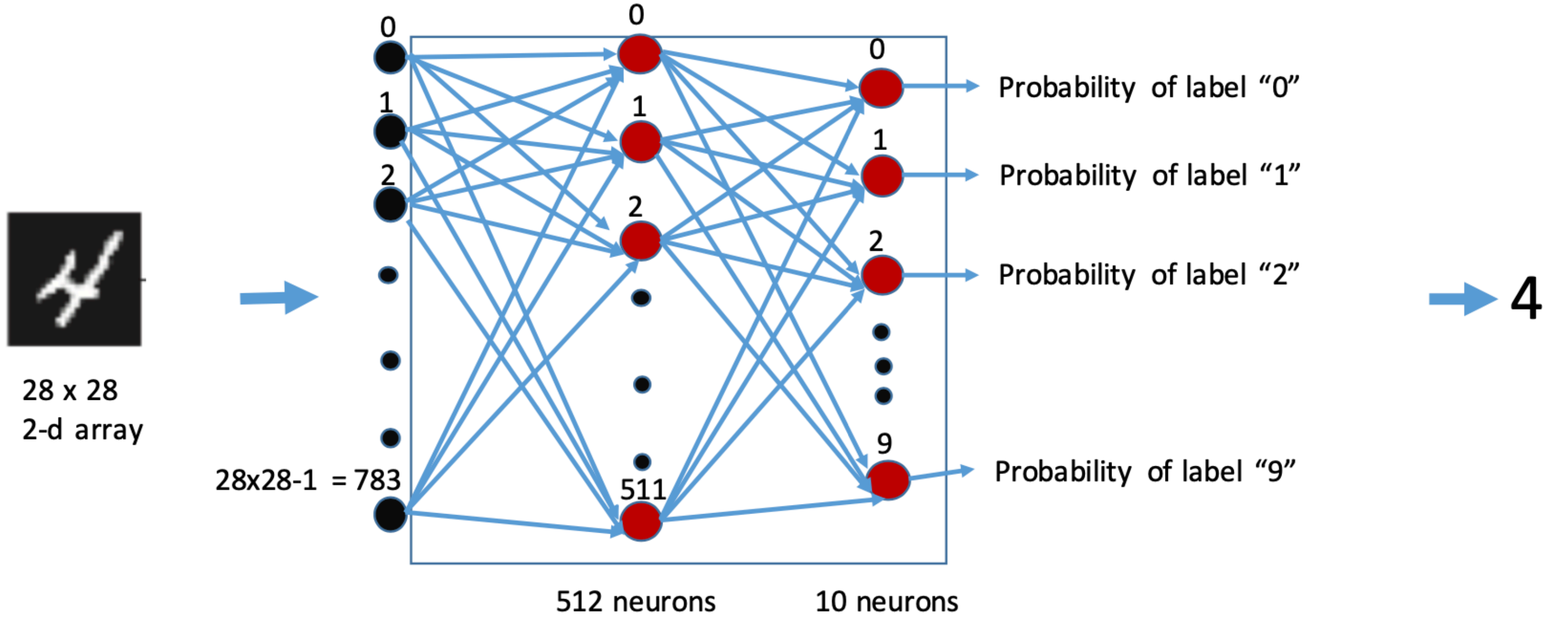


Softmax (an activation function for probabilities)



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'))
```

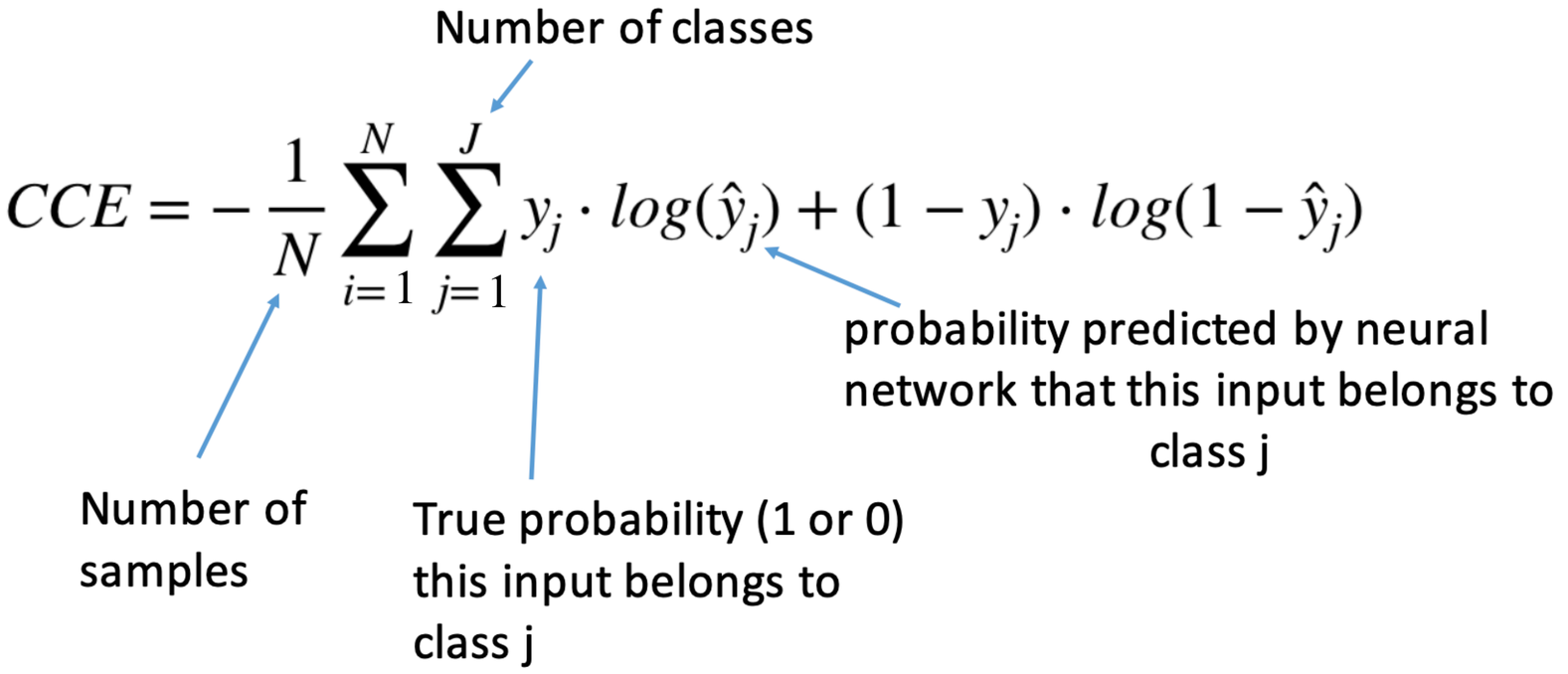


Step 3: choose loss function, optimizer, and target metrics

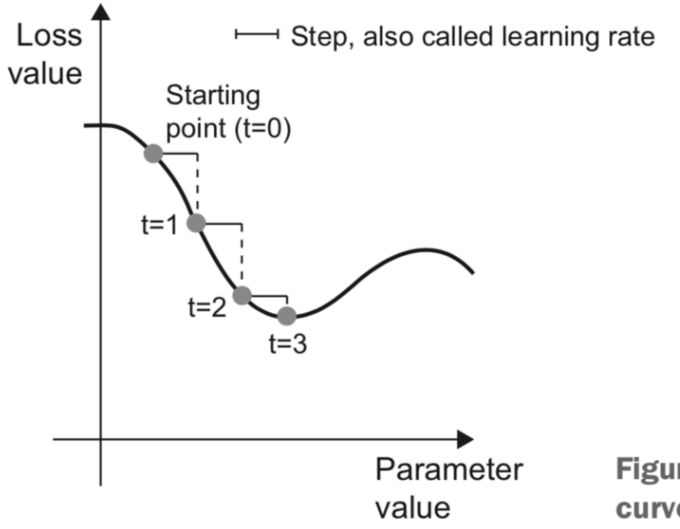
Listing 2.3 The compilation step

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

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



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



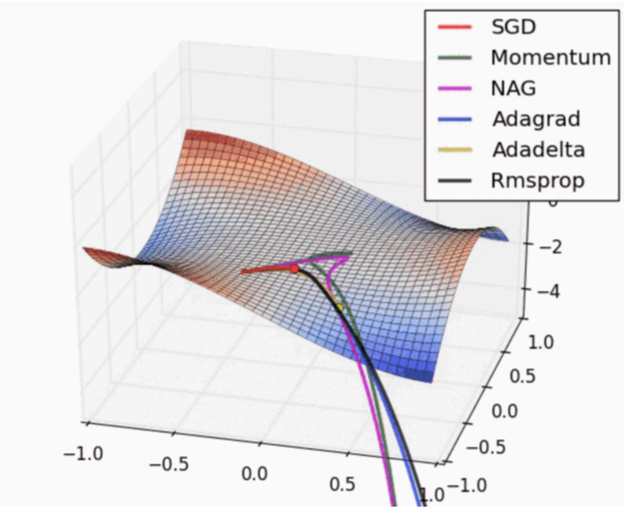
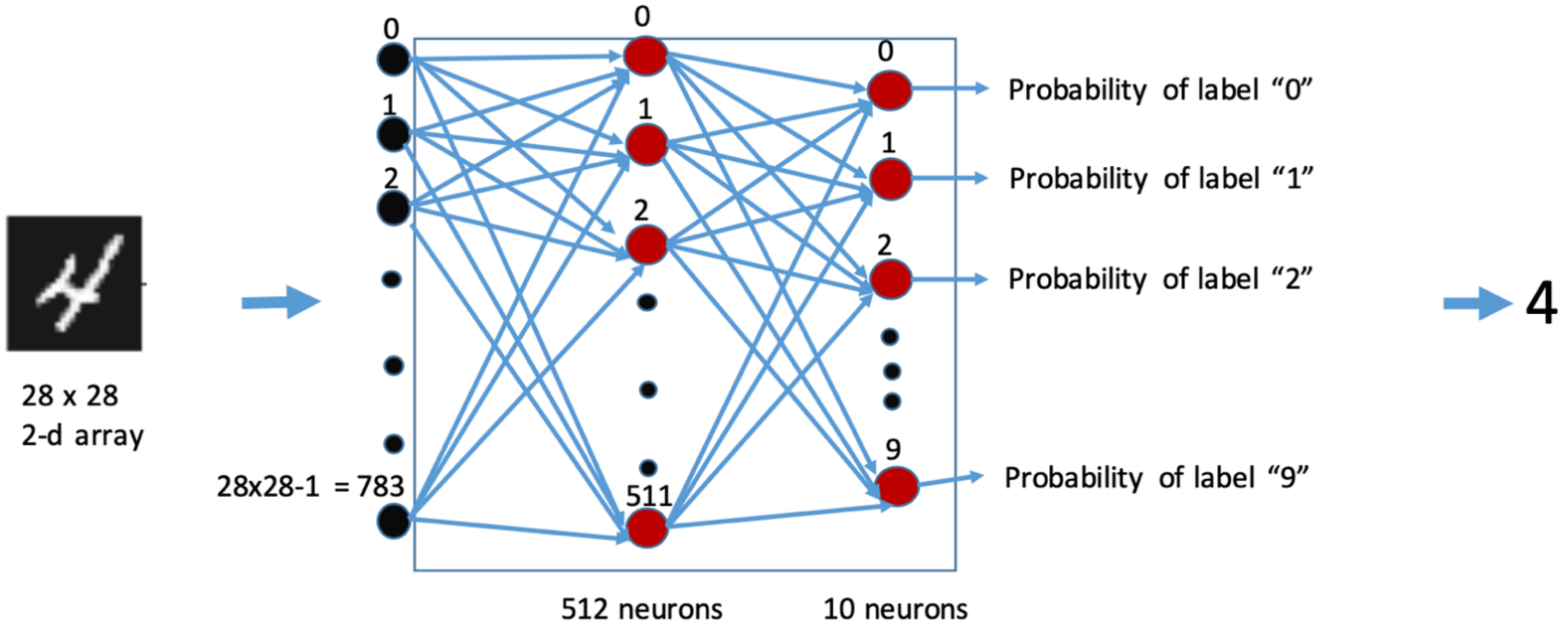


Figure 2.11 SGD down a 1D loss curve (one learnable parameter)

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.

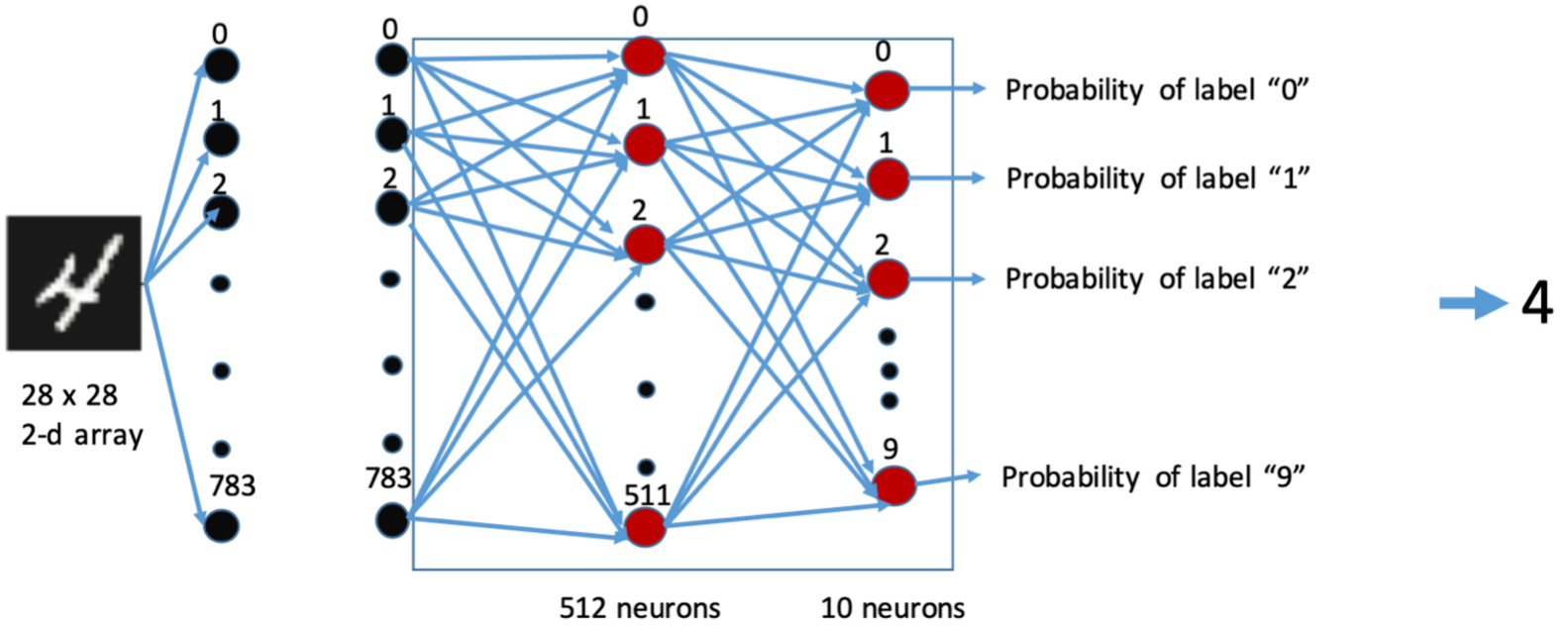

0
1
2
28x28-1 = 783
28 x 28
2-d array
0
1
2
511
512 neurons
0
1
2
9
10 neurons
Probability of label “0”
Probability of label “1”
Probability of label “2”
Probability of label “9”
4

Step 4: Prepare training and test data

Here: Reshape and normalize input training data

Listing 2.4 Preparing the image data

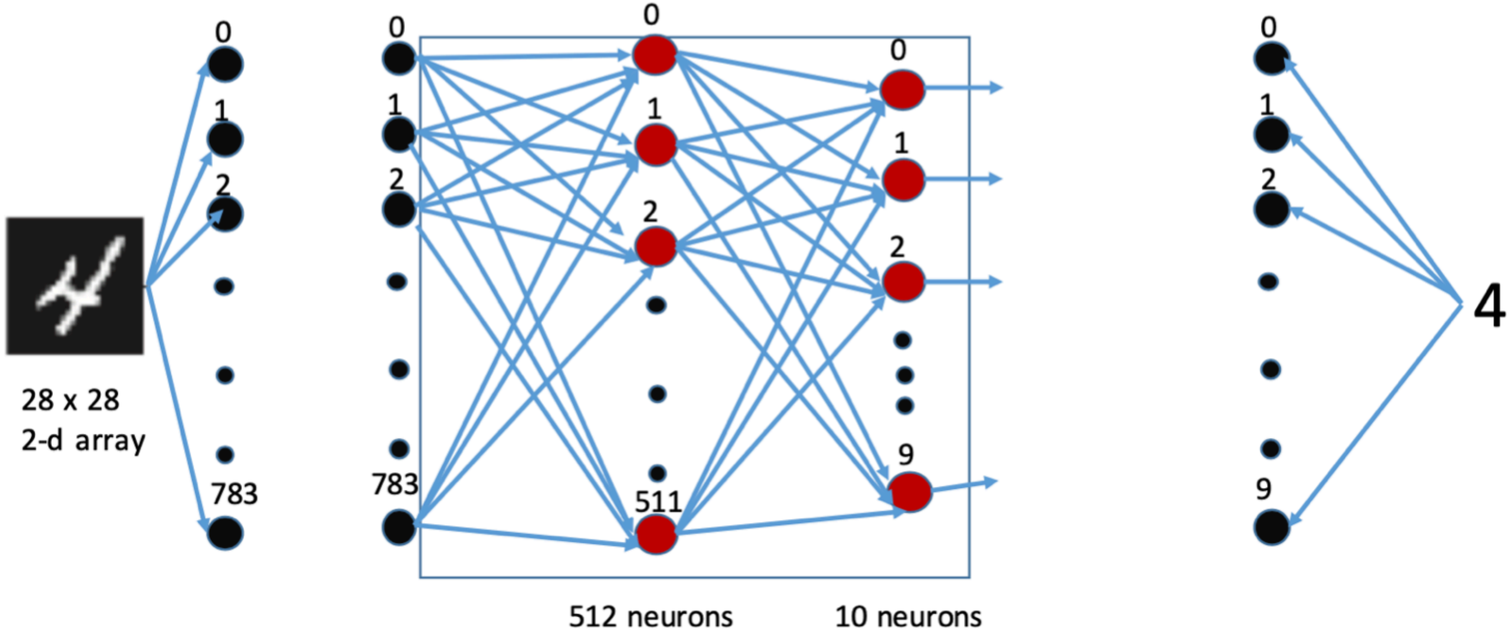
```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

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


0
1
2
783
28 x 28
2-d array
0
1
2
783
0
1
2
511
0
1
2
9
Probability of label “0”
Probability of label “1”
Probability of label “2”
Probability of label “9”
512 neurons
10 neurons
4

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

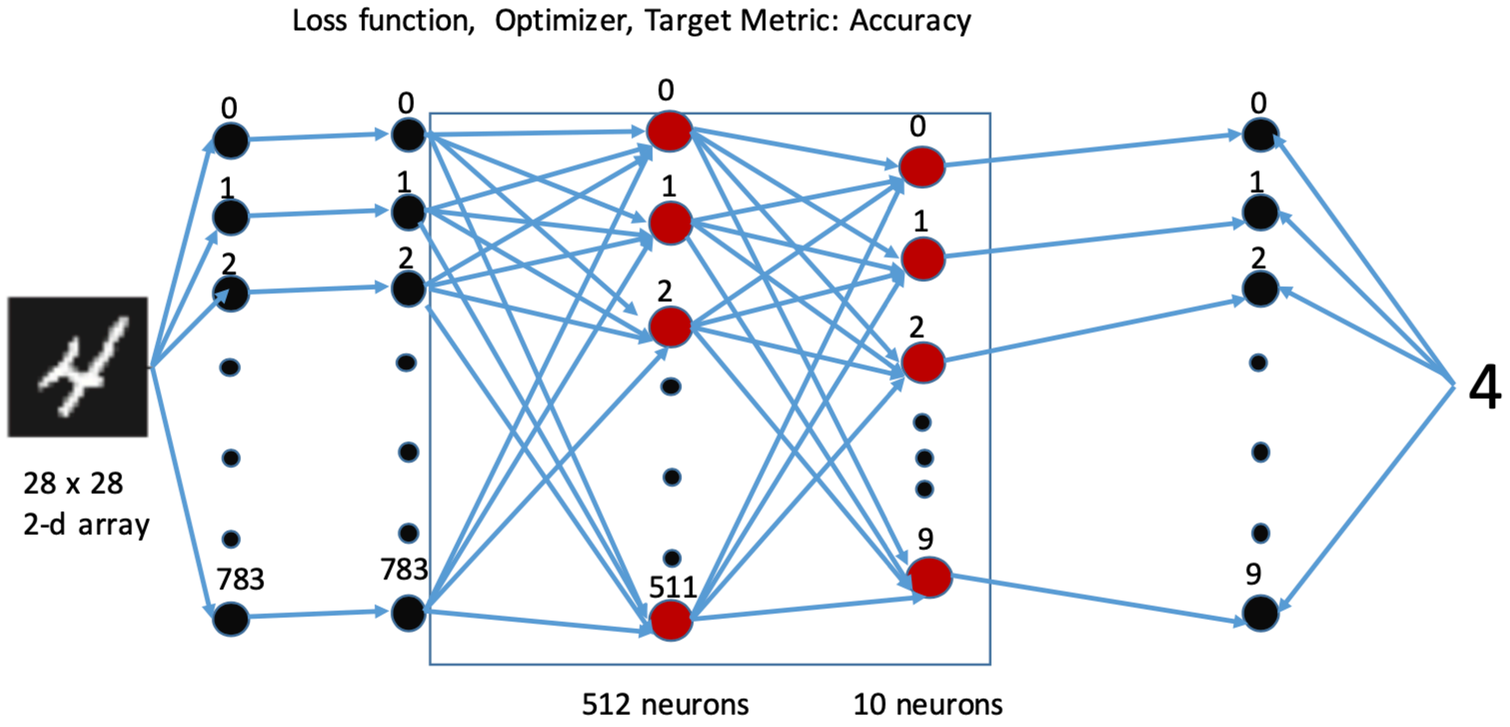
```
from keras.utils import to_categorical

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
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


0
1
2
783
28 x 28
2-d array
0
1
2
783
0
1
2
511
0
1
2
9
512 neurons
10 neurons
0
1
2
9
4

Step 5: Train the neural network

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



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.

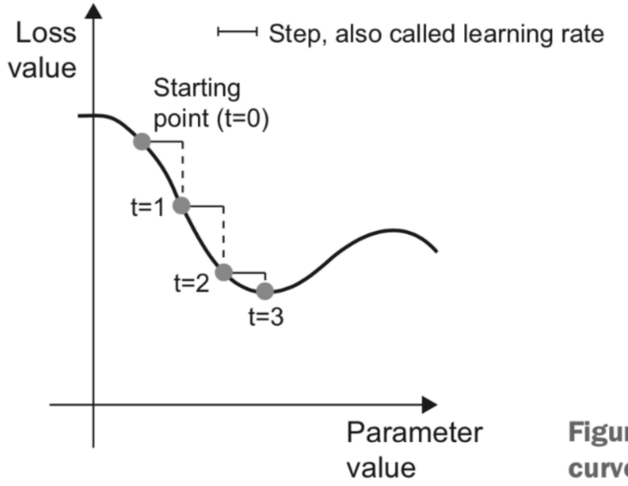


Figure 2.11 SGD down a 1D loss curve (one learnable parameter)

```
>>> 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!

Quiz questions:

1. How to build a sequential neural network?

2. How to prepare data for a neural network?

3. What do “epoch” and “batch size” mean?

Roadmap of this lecture:

1. Handwritten digit recognition.

2. Basic concepts in deep learning.

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)

$A_5^T B_5 = C$ (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

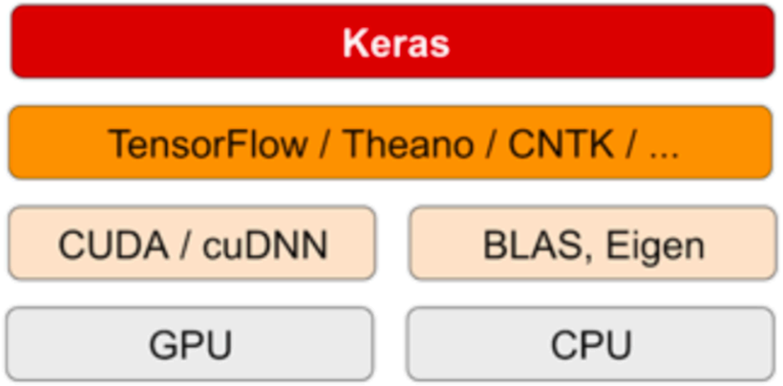
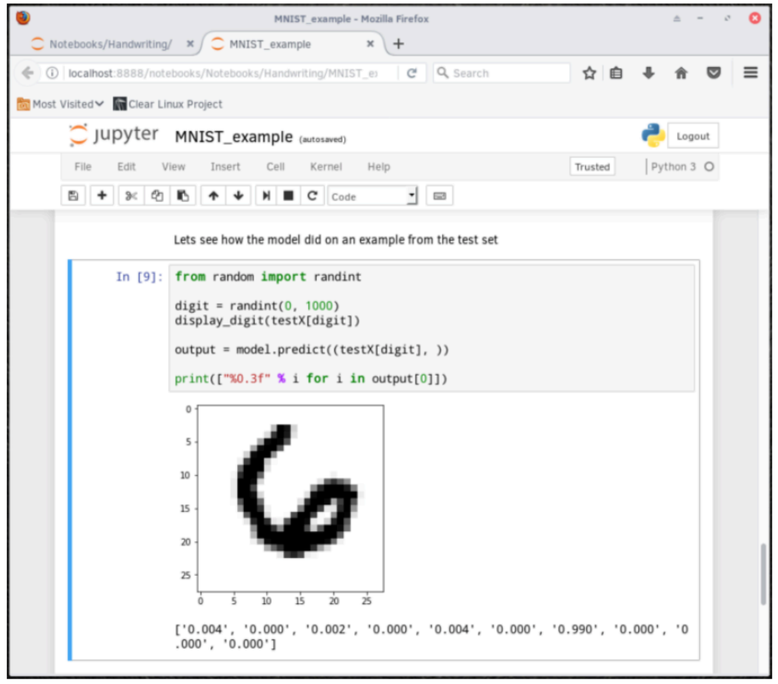


Figure The deep-learning software and hardware stack

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



Tensor: Data representation used in neural network

Rank-0 tensor: scalar number

Rank-1 tensor: vector

Rank-2 tensor: matrix

Rank-3 tensor: 3-dimensional array

Rank-4 tensor: 4-dimensional array

Rank-5 tensor: 5-dimensional array

- *Vector data*—Rank-2 tensors of shape `(samples, features)`, where each sample is a vector of numerical attributes ("features")
- *Timeseries data or sequence data*—Rank-3 tensors of shape `(samples, timesteps, features)`, where each sample is a sequence (of length `timesteps`) of feature vectors
- *Images*—Rank-4 tensors of shape `(samples, height, width, channels)`, where each sample is a 2D grid of pixels, and each pixel is represented by a vector of values ("channels")
- *Video*—Rank-5 tensors of shape `(samples, frames, height, width, channels)`, where each sample is a sequence (of length `frames`) of images

How to train weights: Backpropagation

Automatically compute the gradient of the loss function with respect to the weights in the neural network.

Decrease each weight by:
“learning rate” x “partial derivative of that weight”.

Gradient Tape:

API that can be used to get automatic differentiation

Instantiate a scalar Variable with an initial value of 0.

Open a GradientTape scope.

Inside the scope, apply some tensor operations to our variable.

Use the tape to retrieve the gradient of the output y with respect to our variable x.

```
import tensorflow as tf
x = tf.Variable(0.)
with tf.GradientTape() as tape:
    y = 2 * x + 3
grad_of_y_wrt_x = tape.gradient(y, x)
```

$$y = 2x + 3 \quad \Rightarrow \quad \left.\frac{dy}{dx}\right|_{x=0} = 2$$

```
grad_of_y_wrt_x
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=2.0>
```

Gradient Tape:

API that can be used to get automatic differentiation

```
import tensorflow as tf
x = tf.Variable(3.)
with tf.GradientTape() as tape:
  y = 2 * x * x + 3
grad_of_y_wrt_x = tape.gradient(y,x)
grad_of_y_wrt_x
```

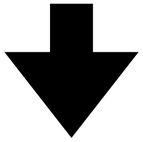
```
<tf.Tensor: shape=(), dtype=float32, numpy=12.0>
```

$$y = 2x^2 + 3 \quad \Rightarrow \quad \left.\frac{dy}{dx}\right|_{x=3} = \left. 4x \right|_{x=3} = 12$$

Gradient Tape works with Tensor operations

```
import tensorflow as tf
x = tf.Variable([[1,2],[3,4]],dtype=tf.float16)
with tf.GradientTape() as tape:
  y = 2 * x + 3
grad_of_y_wrt_x = tape.gradient(y,x)
```

$$x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad y = 2x + 3 = 2\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + 3 = \begin{bmatrix} 5 & 7 \\ 9 & 11 \end{bmatrix}$$



$$\left.\frac{dy}{dx}\right|_{x=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

Gradient Tape works with Tensor operations

```
import tensorflow as tf
x = tf.Variable([[1,2],[3,4]],dtype=tf.float16)
with tf.GradientTape() as tape:
  y = 2 * x * x + 3
grad_of_y_wrt_x = tape.gradient(y,x)
```

$$x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad y = 2x^2 + 3 = \begin{bmatrix} 2 \times 1^2 + 3 & 2 \times 2^2 + 3 \\ 2 \times 3^2 + 3 & 2 \times 4^2 + 3 \end{bmatrix} = \begin{bmatrix} 5 & 11 \\ 21 & 35 \end{bmatrix}$$

$$\Rightarrow \left.\frac{dy}{dx}\right|_{x=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} = \begin{bmatrix} \left.\frac{dy}{dx}\right|_{x=1} & \left.\frac{dy}{dx}\right|_{x=2} \\ \left.\frac{dy}{dx}\right|_{x=3} & \left.\frac{dy}{dx}\right|_{x=4} \end{bmatrix} = \begin{bmatrix} \left.4x\right|_{x=1} & \left.4x\right|_{x=2} \\ \left.4x\right|_{x=3} & \left.4x\right|_{x=4} \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix}$$

Optimizer, loss function, and performance metric

Optimizers:

- `SGD` (with or without momentum)
- `RMSprop`
- `Adam`
- `Adagrad`
- Etc.

Optimizer, loss function, and performance metric

Losses:

- `CategoricalCrossentropy`
- `SparseCategoricalCrossentropy`
- `BinaryCrossentropy`
- `MeanSquaredError`
- `KLDivergence`
- `CosineSimilarity`
- Etc.

Optimizer, loss function, and performance metric

Metrics:

- `CategoricalAccuracy`
- `SparseCategoricalAccuracy`
- `BinaryAccuracy`
- `AUC`
- `Precision`
- `Recall`
- Etc.

Quiz questions:

1. What is a tensor?

2. How to compute the gradient of a function?