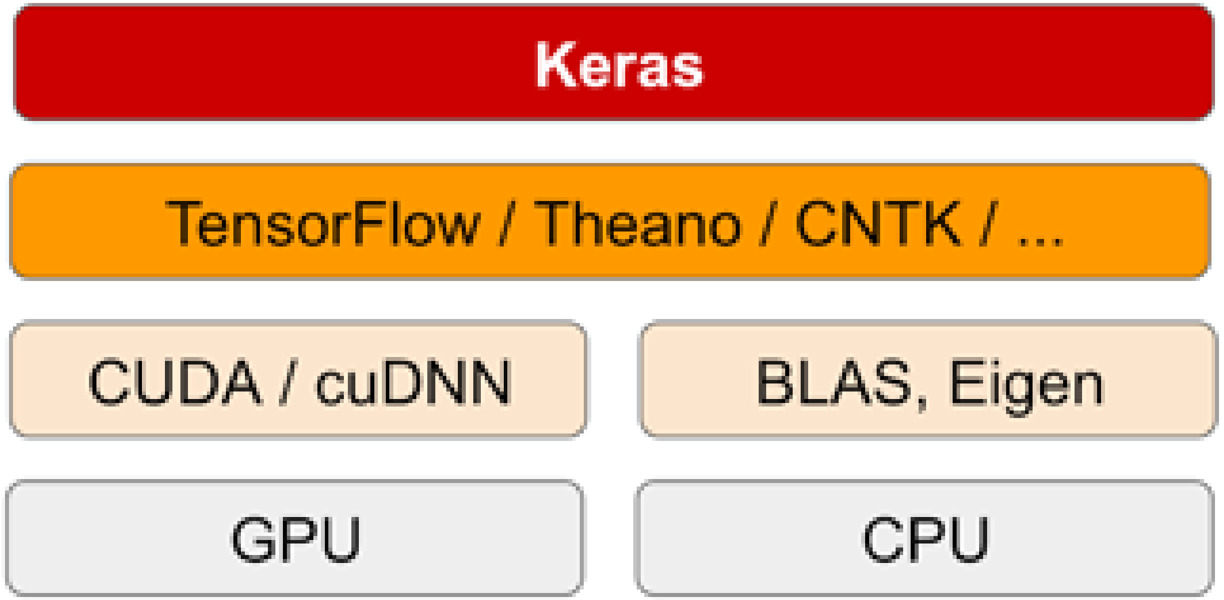
Introduction to Keras

Prof. Kuan-Ting Lai

2021/3/15

Keras (keras.io)

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano
- Developed by Francois Chollet
- Officially supported by TensorFlow 2.0

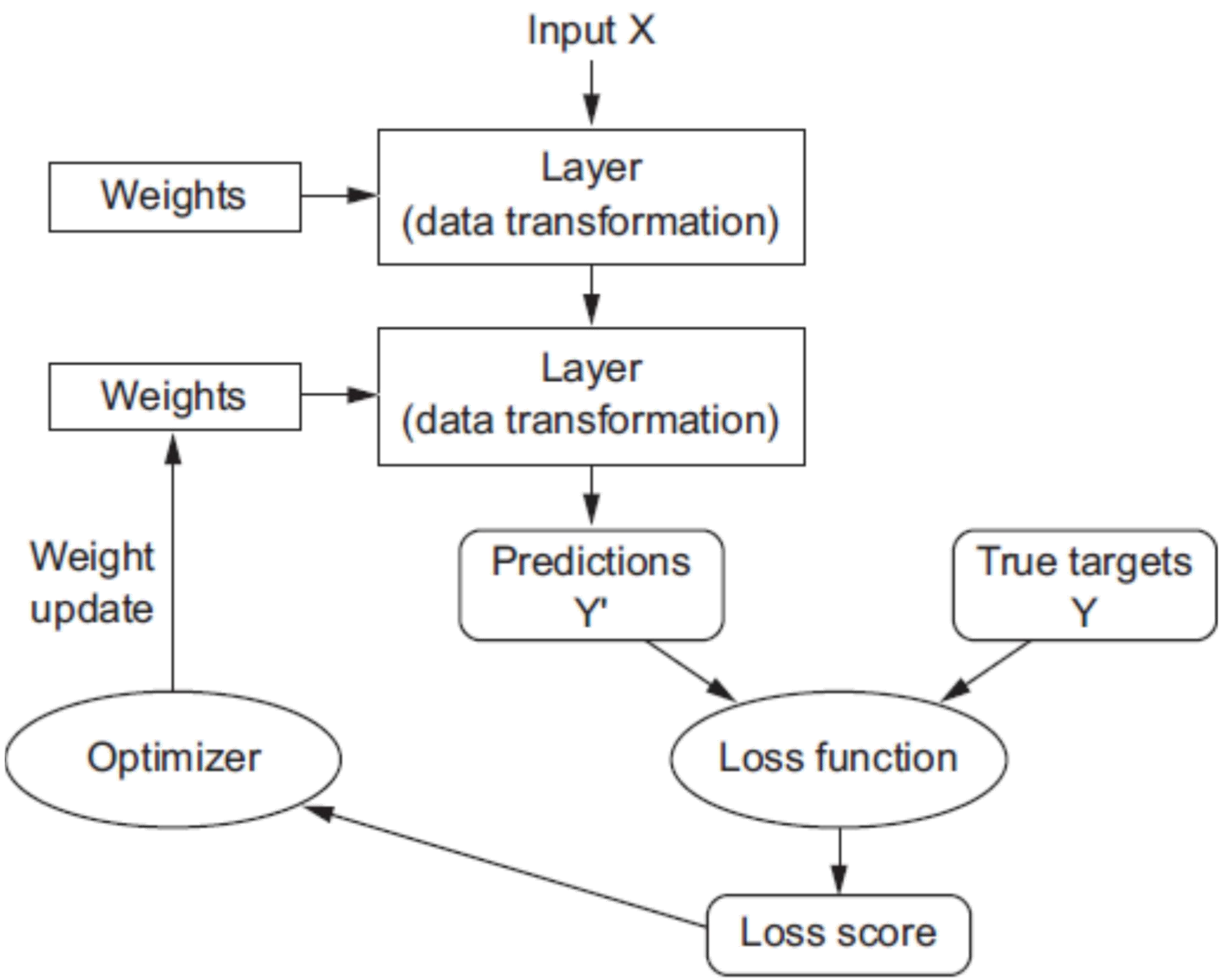


Migrating TensorFlow 1 code to TensorFlow 2

- https://www.tensorflow.org/guide/migrate
- Running 1.X unmodified
  - import tensorflow.compat.v1 as tf
  - tf.disable_v2_behavior()
- Running Keras code
  - Change package "keras" to "tensorflow.keras"
- On Colab
  - Add magic %tensorflow_version 1.x magic

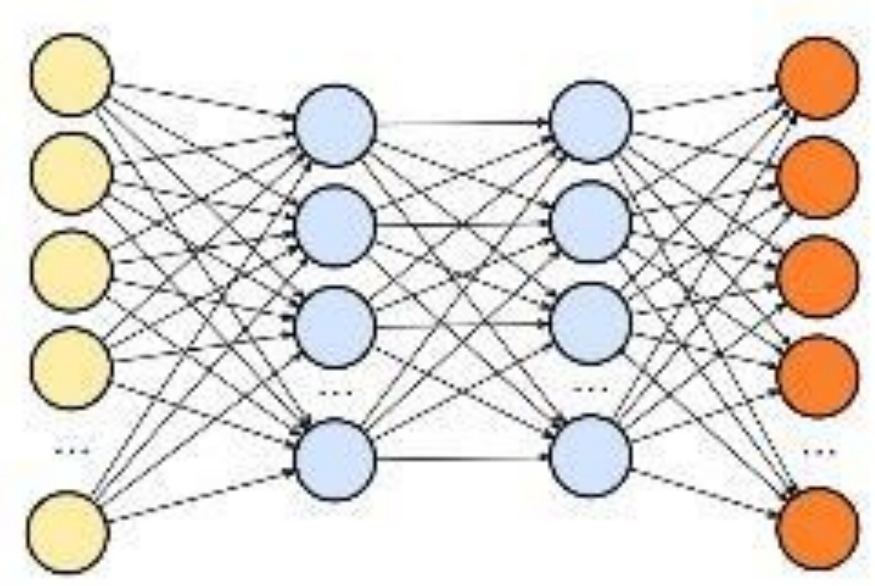
Terminologies of a Neural Network

- Weights
- Layers
- Loss function
- Optimizer



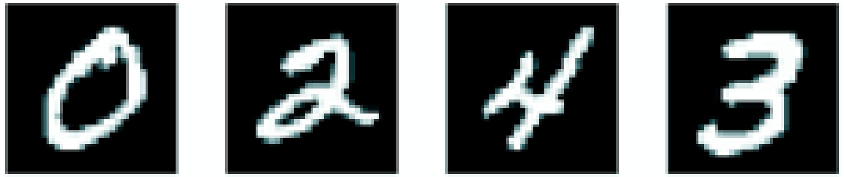
Build Your Own Networks with Keras

- Doing Deep learning with Keras is like playing LEGO



Hello Deep Learning

- Task: classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 ~ 9)
- Use the MNIST dataset created by Yann LeCun
- MNIST has 60,000 training and 10,000 test images



[Colab MNST Code]

Loading MNIST dataset via Keras

```python
from keras.datasets import mnist

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

Loading MNIST via Keras on Colab

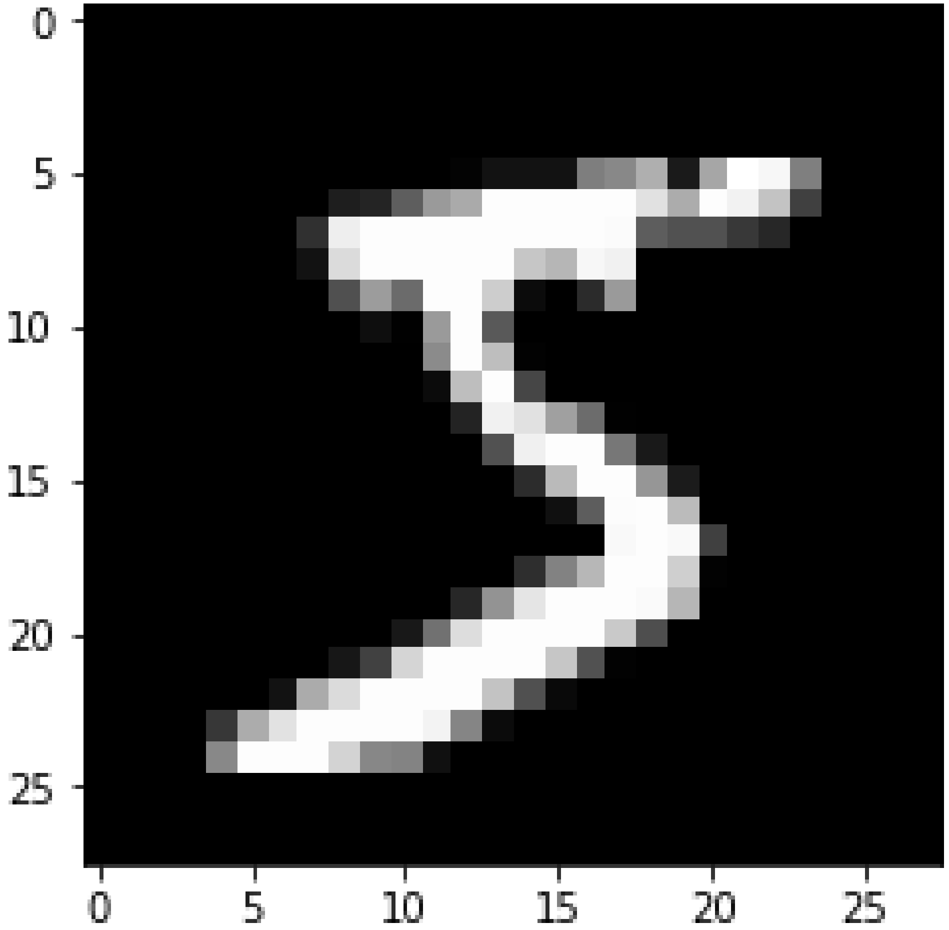
mnist.ipynb

File Edit View Insert Runtime Tools Help

CODE TEXT CELL CELL

```
[2] from keras.datasets import mnist

Using TensorFlow backend.

[3] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step

[4] train_images.shape

(60000, 28, 28)

[5] len(train_labels)

60000

[6] train_labels

array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

[7] test_images.shape

(10000, 28, 28)

[8] len(test_labels)

10000

test_labels

array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

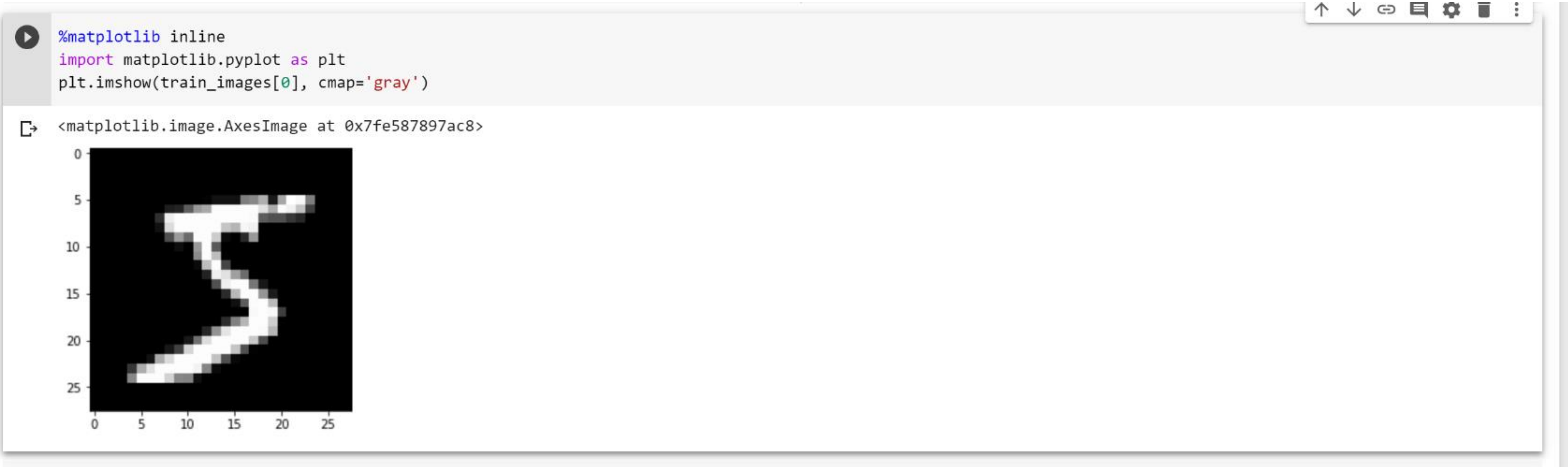
Digital Images

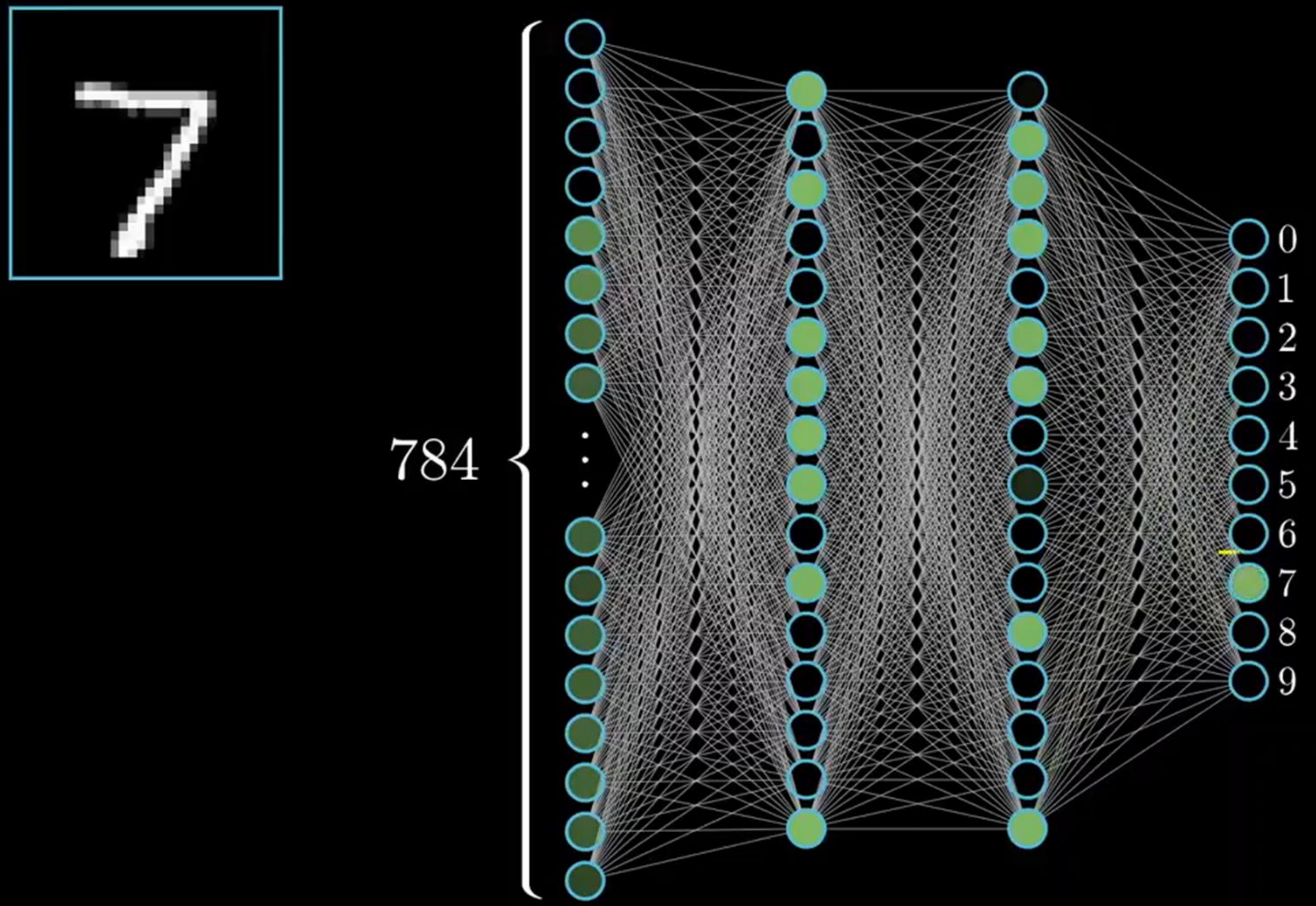



```
array([
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 18, 18, 18, 126, 136, 175, 26, 166, 255, 247,
127, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 30, 36, 94, 154, 170, 253, 253, 253, 253,
253, 225, 172, 253, 242, 195, 64, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 49, 238, 253,
253, 253, 253, 253, 253, 253, 253, 251, 93, 82, 82, 56, 39, 0, 0, 0, 0, 0], [ 0, 0, 0,
0, 0, 0, 0, 18, 219, 253, 253, 253, 253, 253, 198, 182, 247, 241, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 80, 156, 107, 253, 253, 205, 11, 0, 43, 154, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 1, 154, 253, 90, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 139, 253, 190, 2, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 190, 253,
70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35,
241, 225, 160, 108, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 81, 240, 253, 253, 119, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 45, 186, 253, 253, 150, 27, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 93, 252, 253, 187, 0, 0, 0, 0, 0, 0, 0, 0], [ 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 249, 253, 249, 64, 0, 0, 0, 0, 0, 0,
0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 46, 130, 183, 253, 253, 207, 2, 0, 0,
0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 39, 148, 229, 253, 253, 253, 250,
182, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 114, 221, 253, 253,
253, 253, 201, 78, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 23, 66, 213,
253, 253, 253, 253, 198, 81, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 18,
171, 219, 253, 253, 253, 253, 195, 80, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0,
0, 0, 55, 172, 226, 253, 253, 253, 253, 244, 133, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0], [ 0, 0, 0, 0, 136, 253, 253, 253, 212, 135, 132, 16, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]], dtype=uint8)
```

Showing the Images

```
%matplotlib inline

import matplotlib.pyplot as plt
plt.imshow(train_images[0], cmap='gray')
```




784
0
1
2
3
4
5
6
7
8
9

The Network Architecture

- layer: a layer in the deep network for processing data, like a filter
- Dense layer: fully connected neural layer
- Softmax layer: Output probabilities of 10 digits (0 ~ 9)

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(16, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(16, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))
```

Compile Your Model

- Loss function: measure performance on training data
- Optimizer: the mechanism for updating parameters
- Metrics to evaluate the performance on test data (accuracy)

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

Summary of Our Model

```
network.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                12560
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 10)                170
=================================================================
Total params: 13,002
Trainable params: 13,002
Non-trainable params: 0
_________________________________________________________________
```

$16 \times 16 + \underbrace{16}_{\text{bias}} = 272$

Preparing the data & Labels

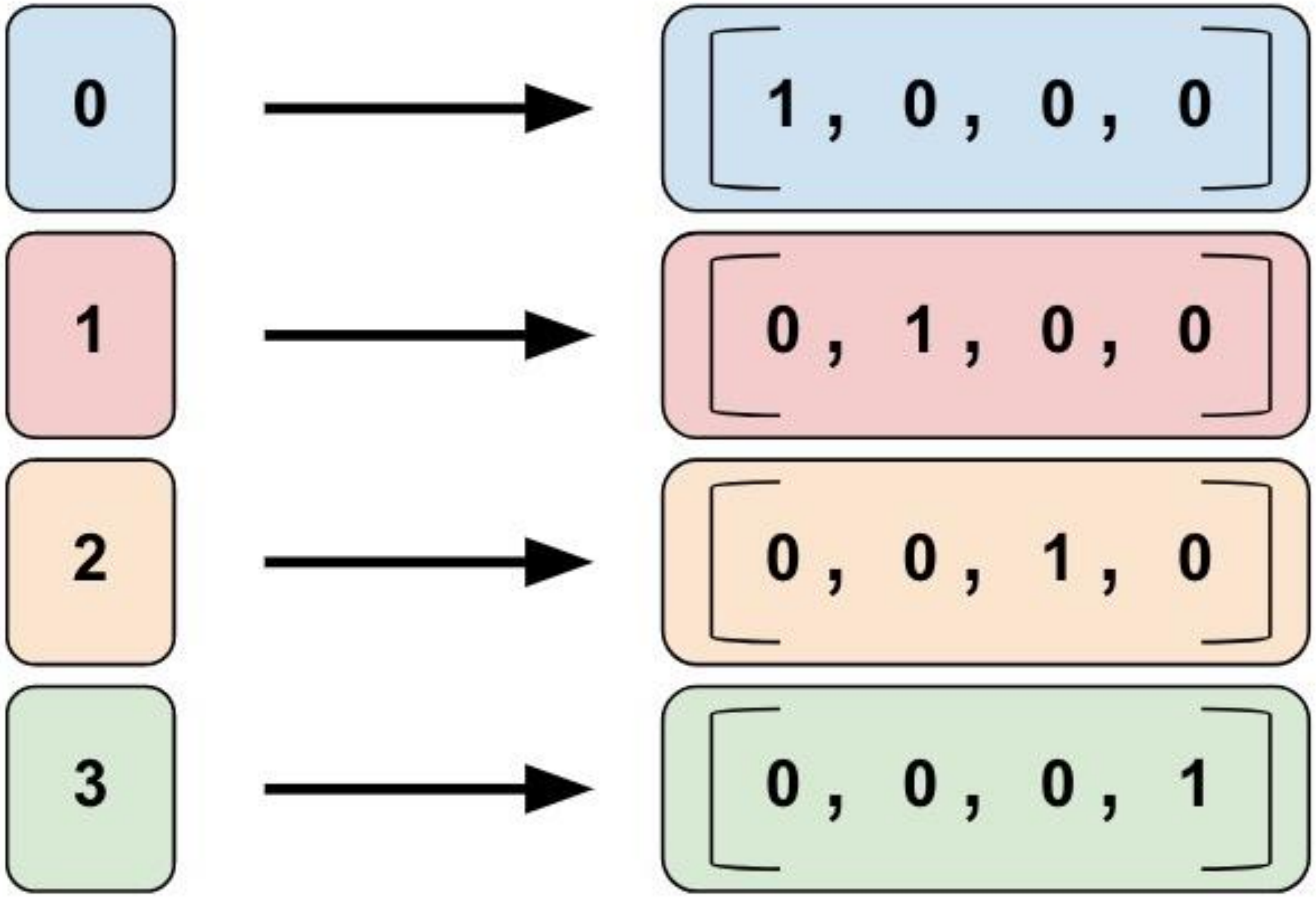
- Preparing the data (Normalization)

```
trn_images = train_images.reshape((60000, 28 * 28))
trn_images = trn_images.astype('float32') / 255
tst_images = test_images.reshape((10000, 28 * 28))
tst_images = tst_images.astype('float32') / 255
```

- Preparing the labels (one-hot encoding)

```
from keras.utils import to_categorical
trn_labels = to_categorical(train_labels)
tst_labels = to_categorical(test_labels)
```

One-hot Encoding



https://tensorflow.rstudio.com/tools/tfdatasets/articles/feature_columns.html

Training

```
network.fit(trn_images, trn_labels, epochs=5, batch_size=128)
```

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

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:306
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
60000/60000 [==============================] - 6s 102us/step - loss: 0.2578 - acc: 0.9249
Epoch 2/5
60000/60000 [==============================] - 6s 94us/step - loss: 0.1026 - acc: 0.9694
Epoch 3/5
60000/60000 [==============================] - 6s 94us/step - loss: 0.0671 - acc: 0.9802
Epoch 4/5
60000/60000 [==============================] - 6s 93us/step - loss: 0.0499 - acc: 0.9850
Epoch 5/5
60000/60000 [==============================] - 6s 93us/step - loss: 0.0376 - acc: 0.9889
<keras.callbacks.History at 0x7f1e5b9c5d68>
```

Complete Code

```
[6] 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'))
```

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

```
[9] trn_images = train_images.reshape((60000, 28 * 28))
    trn_images = trn_images.astype('float32') / 255
    tst_images = test_images.reshape((10000, 28 * 28))
    tst_images = tst_images.astype('float32') / 255
```

```
from keras.utils import to_categorical
trn_labels = to_categorical(train_labels)
tst_labels = to_categorical(test_labels)
```

```
[14] network.fit(trn_images, trn_labels, epochs=5, batch_size=128)
```

Evaluation

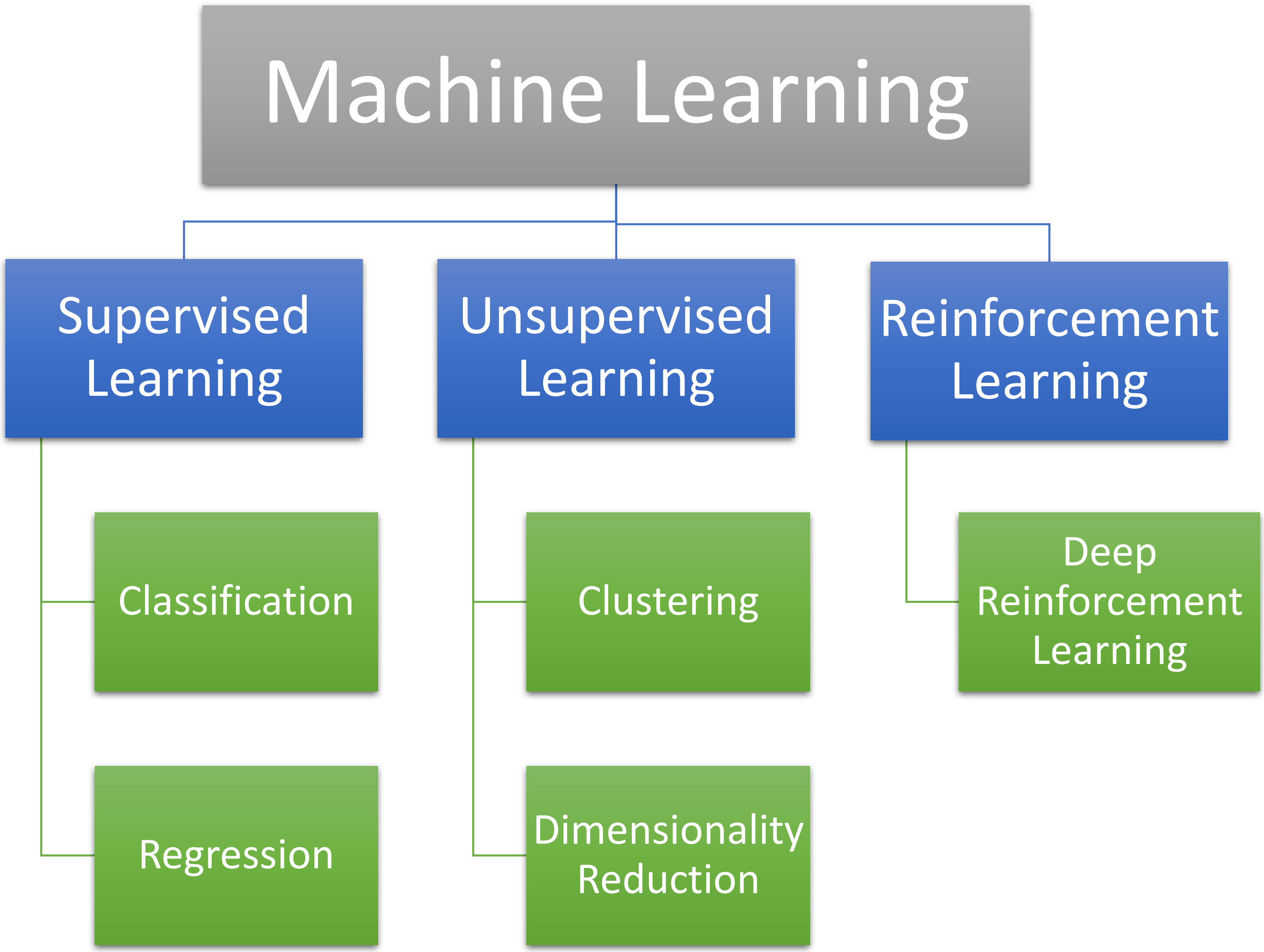
```
test_loss, test_acc = network.evaluate(tst_images, tst_labels)
print('test_acc:', test_acc)
```

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

```
10000/10000 [==============================] - 1s 56us/step
test_acc: 0.9776
```

Classifying Single Input Data

```
[35] probs = network.predict(tst_images[0:1])
     probs

array([[7.2832265e-08, 7.4446991e-09, 1.6131592e-06, 1.7427994e-04,
        4.2739873e-10, 6.5624158e-07, 8.5818193e-12, 9.9980301e-01,
        4.3571268e-07, 1.9945282e-05]], dtype=float32)

[37] import numpy as np
     np.argmax(probs)

7

test_labels[0]

7
```


Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Classification
Regression
Clustering
Dimensionality Reduction
Deep Reinforcement Learning

Deep Learning for Classification & Regression

- Choosing the right last-layer activation and loss function

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

Keras Training Examples

Is the Movie Review Positive?

- Binary Classification
- 50,000 polarized reviews from IMDB

Colab Notebook

Classify Financial News

- Multi-class Classification
- 46 exclusive topics including earn, grain, crude, trade,…

Colab Notebook

Predicting Housing Price

- Regression
- Use Boston housing price dataset with 506 samples and 13 features (crime rate, rooms, age, …)

Colab Notebook

IMDB Movie review dataset (50,000 reviews)

Binary Classification: Check if a Movie Review is Positive or Negative

IMDb Movie Review Dataset

- Internet Movie Database
- 50,000 polarized reviews (50% positive and 50% negative reviews)
- https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset
- Goal
  - Classify if a review is positive or negative (binary classification)

Loading the IMDB dataset

- Packaged in Keras

```python
from keras.datasets import imdb

# num_words is to select the N most frequently used words in all the reviews
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
[ ]  train_data

array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8,
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 534, 19,
       ...,
       list([1, 11, 6, 230, 245, 6401, 9, 6, 1225, 446, 2, 45, 2174, 84, 8322, 4007, 21, 4, 912, 84, 2, 325, 725, 134, 2, 1715, 84, 5, 36, 28, 57, 1099, 21, 8, 140, 8, 703, 5, 2,
       list([1, 1446, 7079, 69, 72, 3305, 13, 610, 930, 8, 12, 582, 23, 5, 16, 484, 685, 54, 349, 11, 4120, 2959, 45, 58, 1466, 13, 197, 12, 16, 43, 23, 2, 5, 62, 30, 145, 402, 1
       list([1, 17, 6, 194, 337, 7, 4, 204, 22, 45, 254, 8, 106, 14, 123, 4, 2, 270, 2, 5, 2, 2, 732, 2098, 101, 405, 39, 14, 1034, 4, 1310, 9, 115, 50, 305, 12, 47, 4, 168, 5, 2
      dtype=object)
```

Decode Data Back to English

```
[ ] word_index = imdb.get_word_index()
    reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

```
Downloading data from https://s3.amazonaws.com/text-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
```

```
# Decodes the review to read the content. Note that the indcies are offset by 3,
# becuase 0, 1, 2 are reserved indices for "padding", "start of sequence," and "Unknown"
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[10]])
decoded_review
```

```
'? french horror cinema has seen something of a revival over the last couple of years with great films such as inside and ? ro
mance ? on to the scene ? ? the revival just slightly but stands head and shoulders over most modern horror titles and is sure
ly one of the best french horror films ever made ? was obviously shot on a low budget but this is made up for in far more ways
than one by the originality of the film and this in turn is ? by the excellent writing and acting that ensure the film is a wi
nner the plot focuses on two main ideas prison and black magic the central character is a man named ? sent to prison for fraud
he is put in a cell with three others the quietly insane ? body building ? marcus and his retarded boyfriend daisy after a sho
rt while in the cell together they stumble upon a hiding place in the wall that contains an old ? after ? part of it they soon
realise its magical powers and realise they may be able to use it to break through the prison walls br br black magi...'
```

Preprocess the Data

- Turn data into tensors
  - Pad the list to make all reviews have the same length
  - Transform integer data into one-hot encoding format

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

Vectorized training data

Vectorized test data

One-hot Encoding of a Review

```
[7]  x_train = vectorize_sequences(train_data)
     x_test = vectorize_sequences(test_data)
```

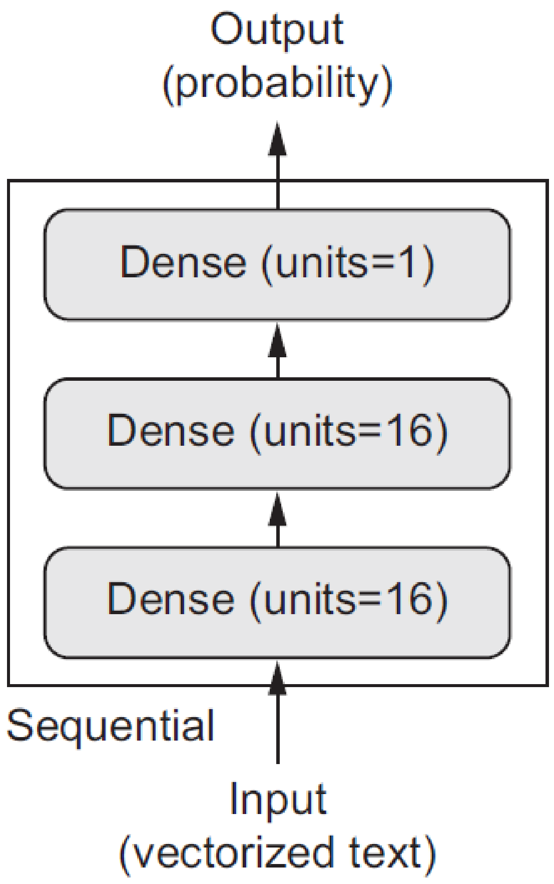
```
[21] x_train[0, 0:100]
```

```
array([0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 1., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 1.,
       0., 1., 1., 0., 1., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 1.,
       1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0.])
```

```
[10] y_train = np.asarray(train_labels).astype('float32')
     y_test = np.asarray(test_labels).astype('float32')
```

Create a three-layer network

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid’))

model.compile(optimizer='rmsprop’,
              loss='binary_crossentropy’,
              metrics=['accuracy'])
```



Select Activation Function

- Select activation function

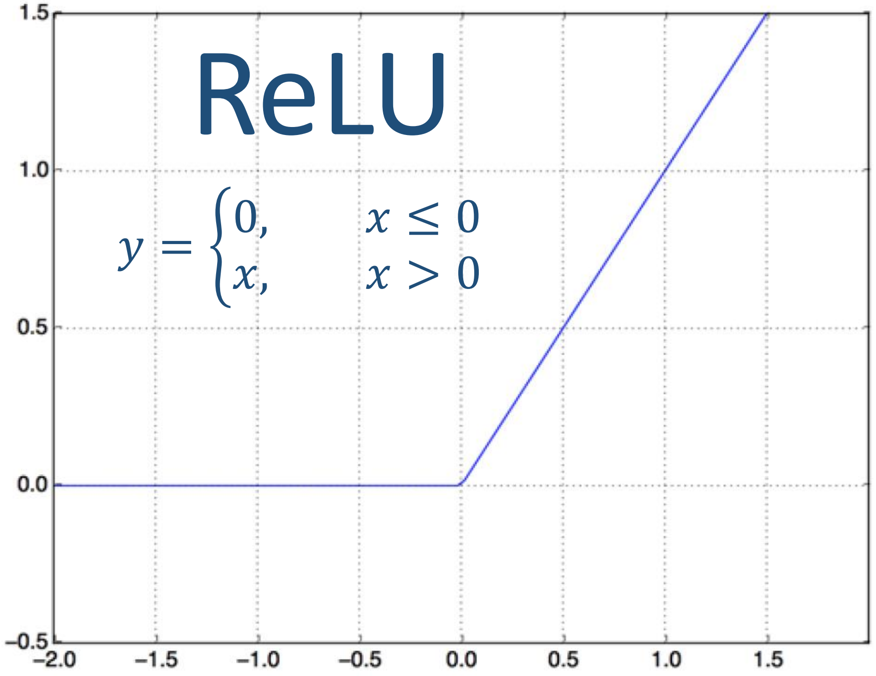


Figure 3.4 The rectified linear unit function

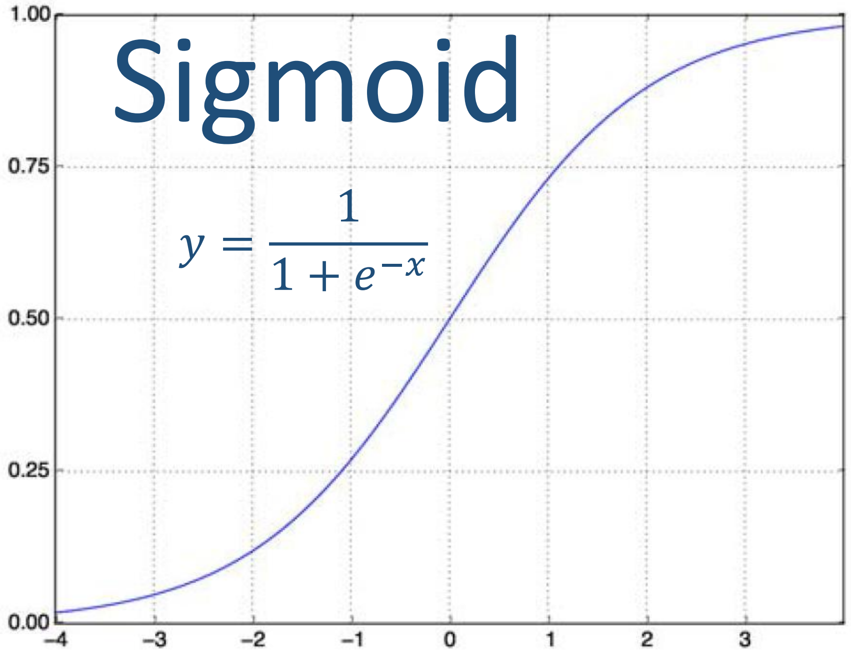


Figure 3.5 The sigmoid function

Why We Need Activation Functions?

- Without an activation function, the Dense layer would consist of two linear operations—a dot product and an addition.
- So the layer could only learn *linear transformations* (affine transformations) of the input data.
- Such a hypothesis space is too restricted and wouldn't benefit from multiple layers of representations.

Customize the Optimizer & Loss & Metric

Listing 3.5 Configuring the optimizer

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Listing 3.6 Using custom losses and metrics

```
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

Split a Validation Set

- Use a separate data to pretend as test data
- Can be used to monitor the model’s accuracy during training
- Select first 10,000 data as validation data

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

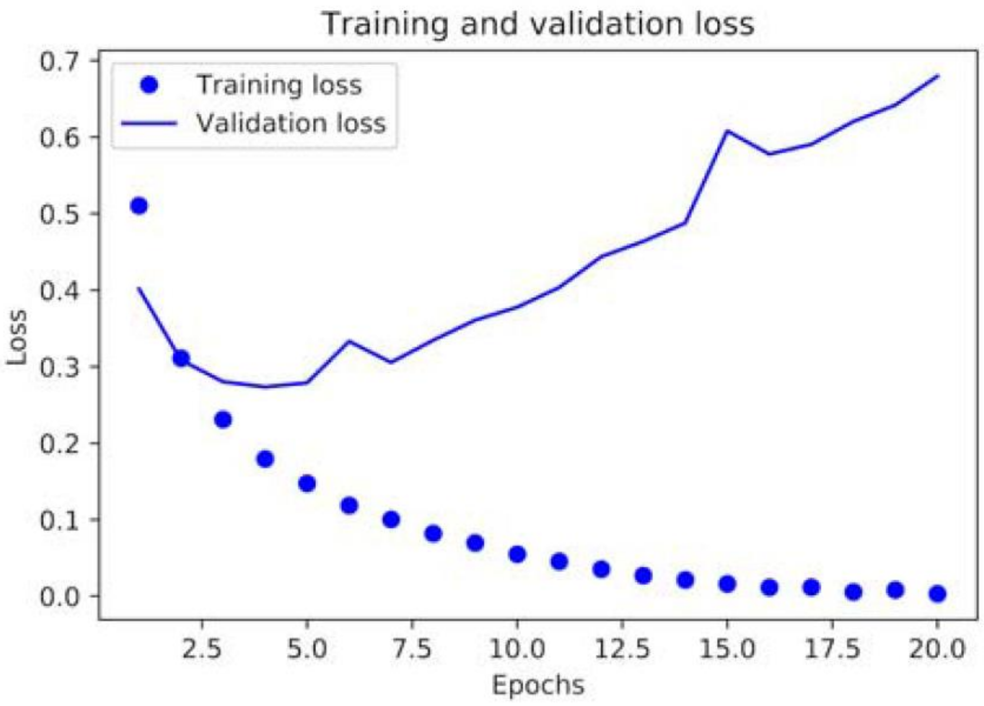
Train the Model

- Batch size = 512
- Epochs = 20

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

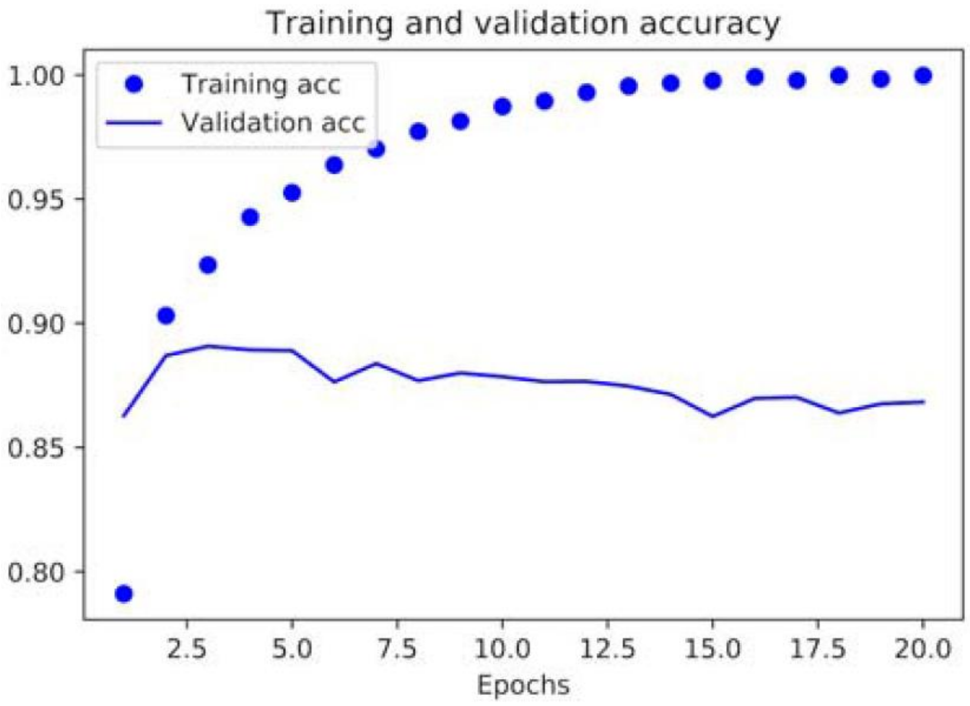
Plot the Training and Validation Loss

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo',
label='Training loss')
plt.plot(epochs, val_loss_values, 'b',
label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Plot the Training and Validation Accuracy

```
acc_values = history_dict[‘accuracy']
val_acc_values = history_dict['val_accuracy']
plt.plot(epochs, acc_values, 'bo',
         label='Training acc')
plt.plot(epochs, val_acc_values, 'b’,
         label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel(‘Accuracy')
plt.legend()
plt.show()
```



Evaluate on Test Data

```
[17] [loss, acc] = model.evaluate(x_test, y_test)
     print('Accuracy: ', acc)
```

```
25000/25000 [==============================] - 2s 74us/step
Accuracy:  0.8442400097846985
```

Use Our Model to Predict

- model.predict()

```
>>> model.predict(x_test)
array([[ 0.98006207]
       [ 0.99758697]
       [ 0.99975556]
       ...,
       [ 0.82167041]
       [ 0.02885115]
       [ 0.65371346]], dtype=float32)
```

Classifying Reuters News Topics (Multi-class Classification)

Reuters Financial News

- A subset of Reuters-21578 dataset from UCI Machine Learning
  – https://archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection
- Single-label, multiclass classification
- 8,982 training and 2,246 testing samples

Load the Reuters Dataset

- Select 10,000 most frequently occurring words

```
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) =
reuters.load_data(num_words=10000)
```

Decode the News

- Decode the word ID list back into English

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in
train_data[0]])
print(decoded_newswire)
```

```
Downloading data from https://s3.amazonaws.com/text-
datasets/reuters_word_index.json 557056/550378
[==============================] - 0s 0us/step ? ? ? said as a result of
its december acquisition of space co it expects earnings per share in 1987
of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said
pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and
rental operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said
cash flow per share this year should be 2 50 to three dlrs reuter 3
```

Perform One-hot Encoding

```
# Encode test data
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1.
  return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

# Encode labels (one-hot encoding)
from keras.utils.np_utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

Split Train/Validation Sets

- Select first 1000 samples as validation set

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

Build Our Model

- Three-layer network
- Note the top is a Softmax function with 46 outputs

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

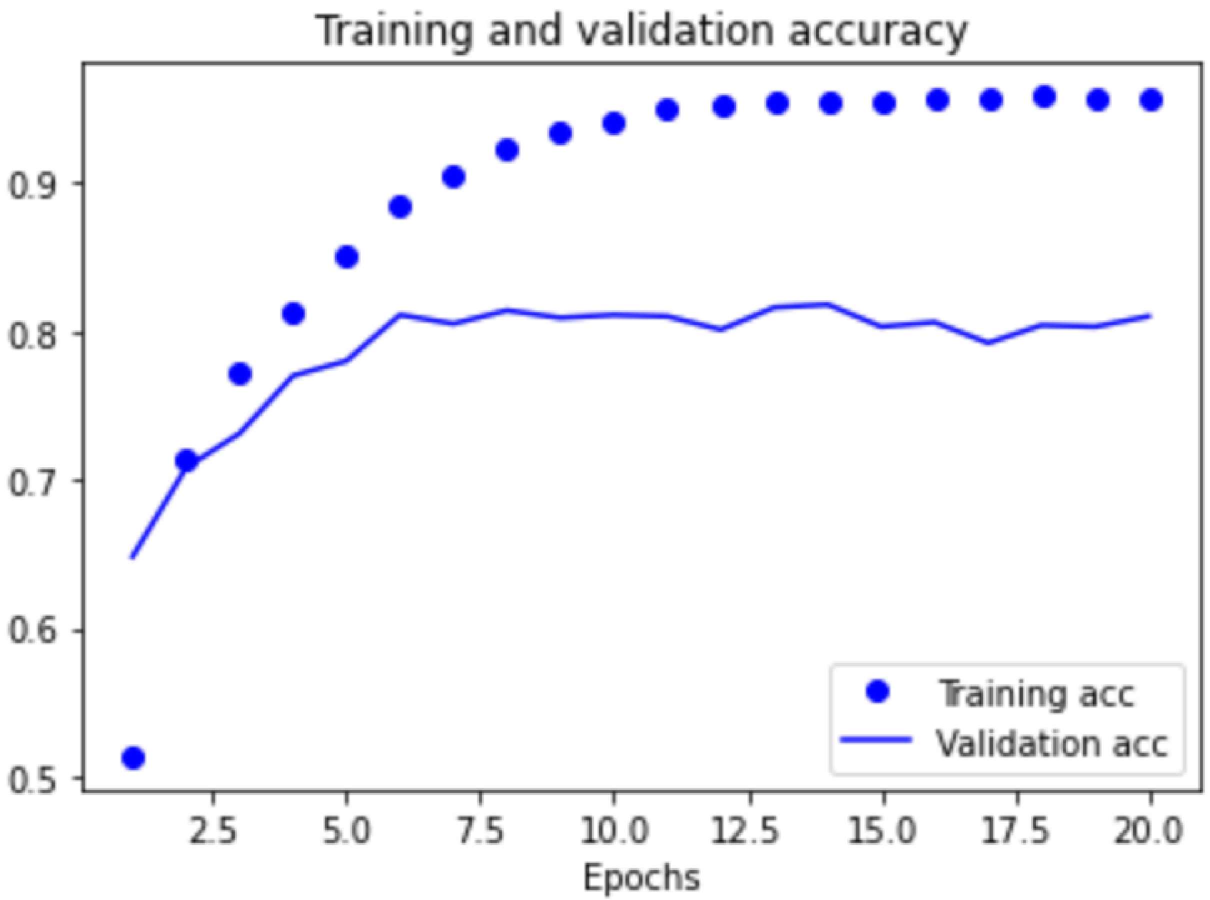
Train our model

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Train on 7982 samples, validate on 1000 samples Epoch 1/20 7982/7982
[==============================] - 1s 120us/step - loss: 2.6697 - acc:
0.5292 - val_loss: 1.7476 - val_acc: 0.6420 Epoch 2/20 7982/7982
[==============================] - 1s 85us/step - loss: 1.4094 - acc:
0.7136 - val_loss: 1.3266 - val_acc: 0.7140 Epoch 3/20 7982/7982
[==============================] - 1s 87us/step - loss: 1.0463 - acc:
0.7757 - val_loss: 1.1496 - val_acc: 0.7470 Epoch 4/20 7982/7982
[==============================] - 1s 87us/step - loss: 0.8318 - acc:
0.8231 - val_loss: 1.0533 - val_acc: 0.7790 Epoch 5/20 7982/7982
[==============================] - 1s 86us/step - loss: 0.6676 - acc:
0.8594 - val_loss: 0.9853 - val_acc: 0.7920 Epoch 6/20 7982/7982
[==============================] - 1s 87us/step - loss: 0.5349 - acc:
0.8870 - val_loss: 0.9367 - val_acc: 0.8070
……
```

Plot Training Accuracy vs. Validation Accuracy

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training
acc')
plt.plot(epochs, val_acc, 'b',
label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Test Our Model

- Achieved around 76.93% accuracy on 2246 test samples

```
results = model.evaluate(x_test, one_hot_test_labels)
print(results)
print(model.metrics_names)
```

```
2246/2246 [==============================] - 0s 98us/step
[1.271177417554711, 0.7693677649684815] ['loss', 'acc']
```

Different Ways to Handle Labels and Loss

- Use integer labels

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

- Select the loss function (sparse_categorical_crossentropy)

```
model.compile(optimizer='rmsprop',
loss='sparse_categorical_crossentropy',
metrics=['acc'])
```

Summary of Multi-class Classification

- To classify *N* classes, the output layer's size should be *N*.
- In a single-label, multiclass classification problem, the output layer should choose a Softmax activation with *N* output classes.
- Categorical cross entropy is the go-to loss function for classification problems
- There are two ways to handle labels in multiclass classification:
  - One-hot encoding + categorical_crossentropy
  - labels encoding (as integers) + sparse_categorical_crossentropy

Predicting Boston House Prices (Regression Example)

Boston Housing Price Dataset

- Goal: predict the median price of homes
- Small dataset with 506 samples and 13 features
  - https://www.kaggle.com/c/boston-housing

1	crime	per capita crime rate by town.	8	dis	weighted mean of distances to five Boston employment centres.
2	zn	proportion of residential land zoned for lots over 25,000 sq.ft.	9	rad	index of accessibility to radial highways.
3	indus	proportion of non-retail business acres per town.	10	tax	full-value property-tax rate per $10,000.
4	chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).	11	ptratio	pupil-teacher ratio by town.
5	nox	nitrogen oxides concentration	12	black	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
6	rm	average number of rooms per dwelling.	13	lstat	lower status of the population (percent).
7	age	proportion of owner-occupied units built prior to 1940.			median value of owner-occupied homes in $1000

Load the Dataset

- Load from Keras built-in datasets

```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()
```

Normalize the Data

- Make all the feature center around 0 and has a unit standard deviation
- Note that the quantities (mean, std) used for normalizing the test data are computed using the training data!

```
# Normalize the data
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

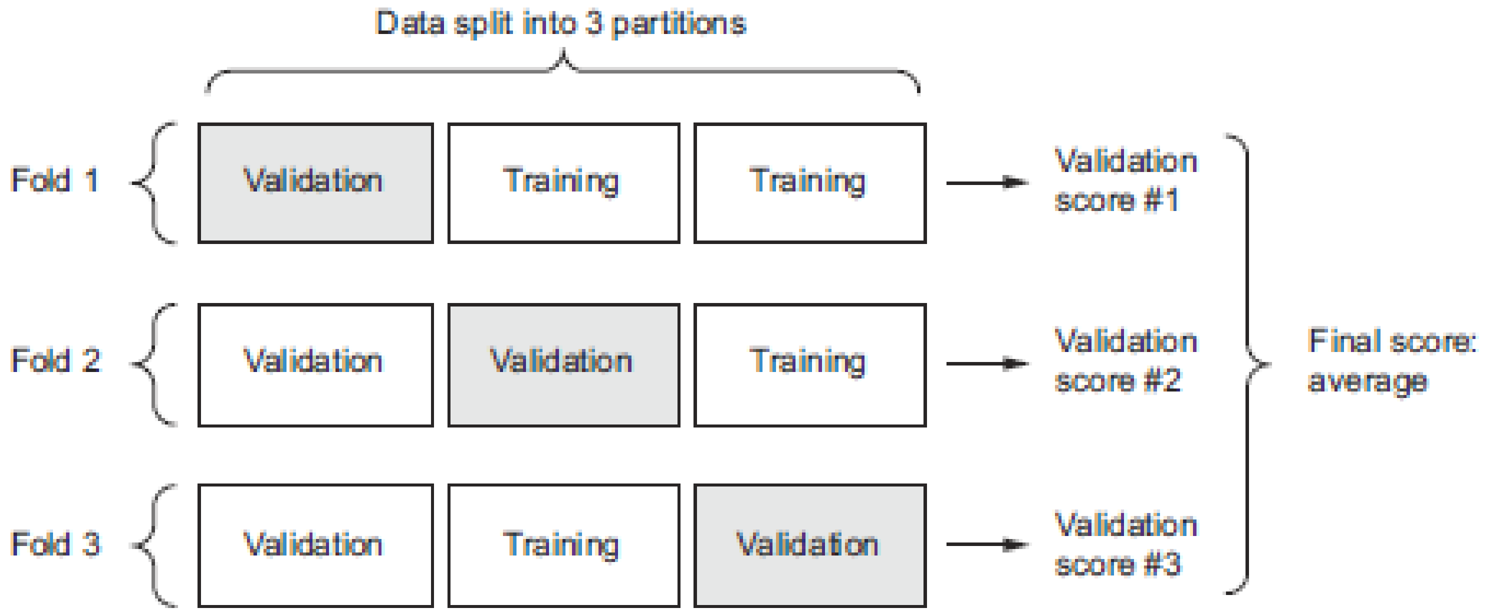
Build the Model

- The network ends with a single unit and no activation
- Loss function: Mean-Squared Error (mse)
- Metrics: Mean Absolute Error (MAE)

```
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
       input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

Cross Validation

- Lower the variance of validation set
- Example: three-fold validation



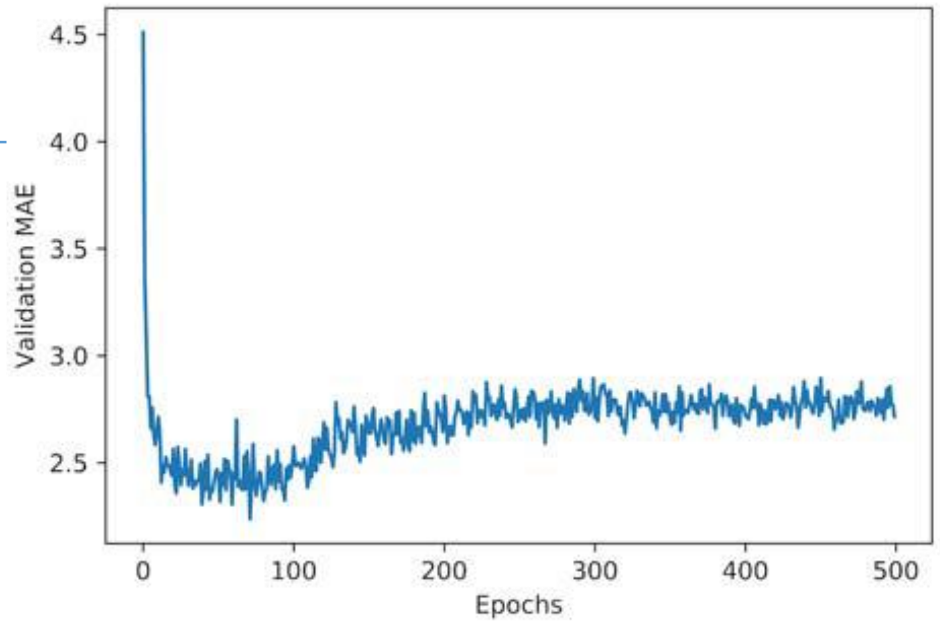
Implement K-fold Validation

```
import numpy as np
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
        train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
        train_targets[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        epochs=num_epochs, batch_size=1, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)

    # For visualization
    mae_history = history.history['mae']
    all_mae_histories.append(mae_history)
```

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

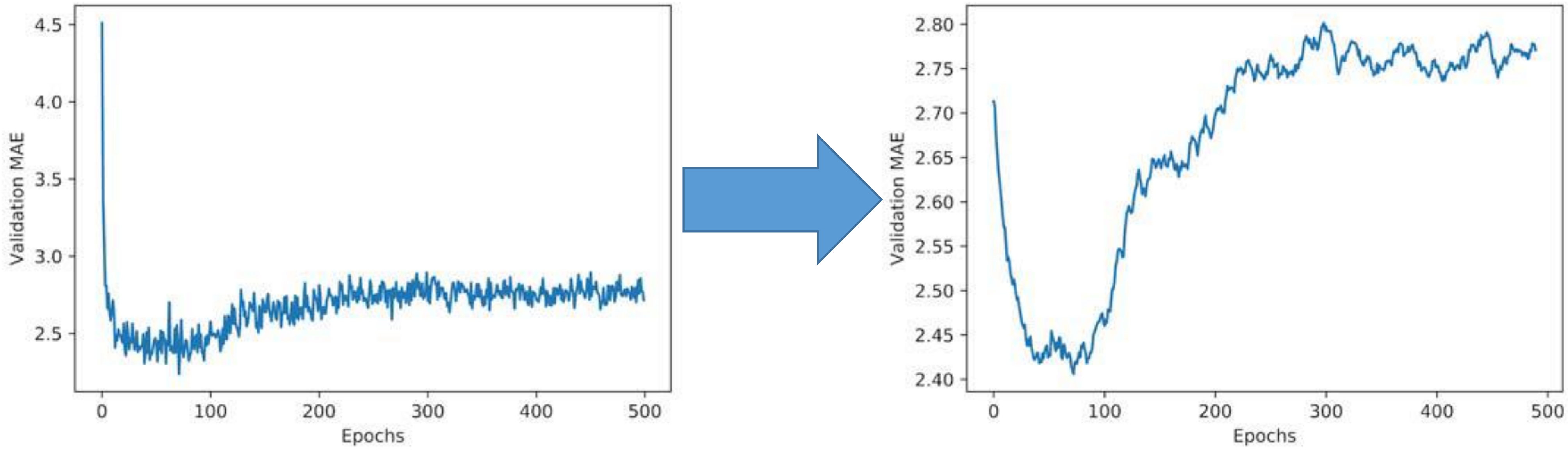
Visualize the averaged MAE scores

```
average_mae_history = [
np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]

import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



Smooth the MAE Scores

- Omit the first 10 data points, which are on a different scale than the rest of the curve.
- Replace each point with an exponential moving average of the previous points, to obtain a smooth curve.



Smooth the Data

```
def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])
```

Train the Final Model

- Train a final production model on all of the training data, with the best parameters

```
model = build_model()

model.fit(train_data, train_targets, epochs=80, batch_size=16, verbose=0)

test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

Evaluate the Final Model

```
[13] test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
     print('The Mean Absolute Error on Testing data is {}'.format(test_mae_score))
```

```
4/4 [==============================] - 0s 3ms/step - loss: 18.6615 - mae: 2.7883
The Mean Absolute Error on Testing data is 2.788318157196045
```

```
[24] np.mean(test_targets)
```

```
23.07843137254902
```

```
[14] model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 64)                896
_________________________________________________________________
dense_16 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 65
=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```

Summary

- The final output layer of a regression model has no activation function
- Use Mean-Squared Error (MSE) as loss function and mean absolute error (MAE) as metric.
- Data need to be normalized
- Use K-fold validation is a great way to reliably evaluate a model.
- Use small network for small training data

Key Takeaways of Today's Class

- `from keras import models, layers`
- `models.Sequential(), add(), compile(), fit(), evaluate()`

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

References

- Francois Chollet, “Deep Learning with Python”, Chapter 3
- https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/