Keras workflow

- Outline
  - Define training data: input tensors and target tensors
  - Define a model: network of layers that maps inputs to targets
  - Configure the learning process (loss function, optimizer, and metrics to monitor)
  - Train using fit() and search hyper-parameters

- Define a model
  - Model types
    - Sequential class - linear stacks of layers (most common)
    - Functional API - supports arbitrary architectures (e.g. DAG). Specify function-like layers to process the data
  - The remaining steps do not depend on model architecture
  - Example:

```
#Sequential class example
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))

#Corresponding functional API example:
from keras import models
from keras import layers
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

Keras workflow

- Compile the network
  - Configure the learning process:
    - optimizer
    - loss function(s)
    - metrics to monitor during training
  - Example:

```
from keras import optimizers

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

- Learning
  - Fit using Numpy arrays of input data and the corresponding target data
  - Example:

```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

Basic example

```
#
# Basic Keras example (MNIST)
#
import numpy as np
import keras
keras.__version__

# Load data
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images.shape # (60000, 28, 28)
len(train_labels)  # 60000
train_labels       # array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

test_images.shape  # (10000, 28, 28)
len(test_labels)   # 10000
test_labels        # array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)

# Reshape and scale 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
```

Basic example

```
# Prepare labels (one hot encoding)
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
train_labels
  # array([[0., 0., 0., ..., 0., 0., 0.],
  #        [1., 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., 1., 0.]], dtype=float32)

test_labels = to_categorical(test_labels)
test_labels
  # array([[0., 0., 0., ..., 1., 0., 0.],
  #       [0., 0., 1., ..., 0., 0., 0.],
  #       [0., 1., 0., ..., 0., 0., 0.],
  #       ...,
  #       [0., 0., 0., ..., 0., 0., 0.],
  #       [0., 0., 0., ..., 0., 0., 0.],
  #       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

Basic example

```
# Build network
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'))
network.summary()
  # Model: "sequential_3"
  # _________________________________________________________________
  # Layer (type)                 Output Shape              Param #
  # =================================================================
  # dense_6 (Dense)              (None, 512)               401920
  # _________________________________________________________________
  # dense_7 (Dense)              (None, 10)                5130
  # =================================================================
  # Total params: 407,050
  # Trainable params: 407,050
  # Non-trainable params: 0

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

$512 \times (28 \times 28 + 1)$

$10 \times (512 + 1)$

Basic example

```
# Train network (with optional validation)
history = network.fit(train_images, train_labels, epochs=5, batch_size=128,
                      validation_data=(test_images, test_labels))
  # Epoch 1/5
  # 469/469 [==============================] - 5s 10ms/step - loss: 0.2536 -
accuracy: 0.9268 -
  #                                           val_loss: 0.1464 - val_accuracy:
0.9559
  # Epoch 2/5
  # 469/469 [==============================] - 4s 9ms/step - loss: 0.1015 -
accuracy: 0.9699 -
  #                                           val_loss: 0.0820 - val_accuracy:
0.9739
  # Epoch 3/5
  # 469/469 [==============================] - 4s 10ms/step - loss: 0.0667 -
accuracy: 0.9802 -
  #                                           val_loss: 0.0723 - val_accuracy:


  # Epoch 5/5
  # 469/469 [==============================] - 4s 9ms/step - loss: 0.0365 -
accuracy: 0.9891 -
  #                                           val_loss: 0.0686 - val_accuracy:
0.9794
```

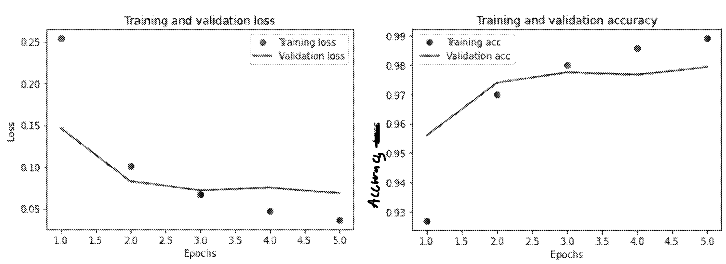
Basic example

```
# Plot accuracy and loss
import matplotlib.pyplot as plt
history_dict = history.history
history_dict.keys() # dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss') # "bo" = "blue dot"
plt.plot(epochs, val_loss, 'b', label='Validation loss') # b = "blue line"
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.clf()
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()
```

Basic example



Basic example

```
# Separate evaluation (if not done during training)
test_loss, test_acc = network.evaluate(test_images, test_labels)
  # 313/313 [==============================] - 1s 2ms/step - loss: 0.0626 -
accuracy: 0.9805

# Inference
test_results = network.predict(test_images)
test_results
  # array([[3.72965481e-09, 2.25926430e-10, 1.78893754e-06, ...,
  #          9.99929309e-01, 5.13594820e-08, 1.71254271e-07],
  #         [6.57615740e-10, 7.97380693e-04, 9.99201477e-01, ...,
  #          1.12863000e-15, 9.58225215e-08, 6.21968458e-15],
  #         [1.34827374e-07, 9.99758303e-01, 1.85805384e-05, ...,
  #          1.59725256e-04, 3.83623301e-05, 5.09398376e-07],
  #         ...,
predictions = np.argmax(test_results, axis=1)
predictions
  # array([7, 2, 1, ..., 4, 5, 6])
confidence = np.max(test_results, axis=1)
confidence
  # array([0.9999293 , 0.9992015 , 0.9997583 , ..., 0.9999949 , 0.99997365,
  #         1.        ], dtype=float32)
```