Chapter 3:

# Neural networks

# Outlines

1. Anatomy of neural networks
2. Introduction to Keras
3. Keras in practice

# Anatomy of neural networks

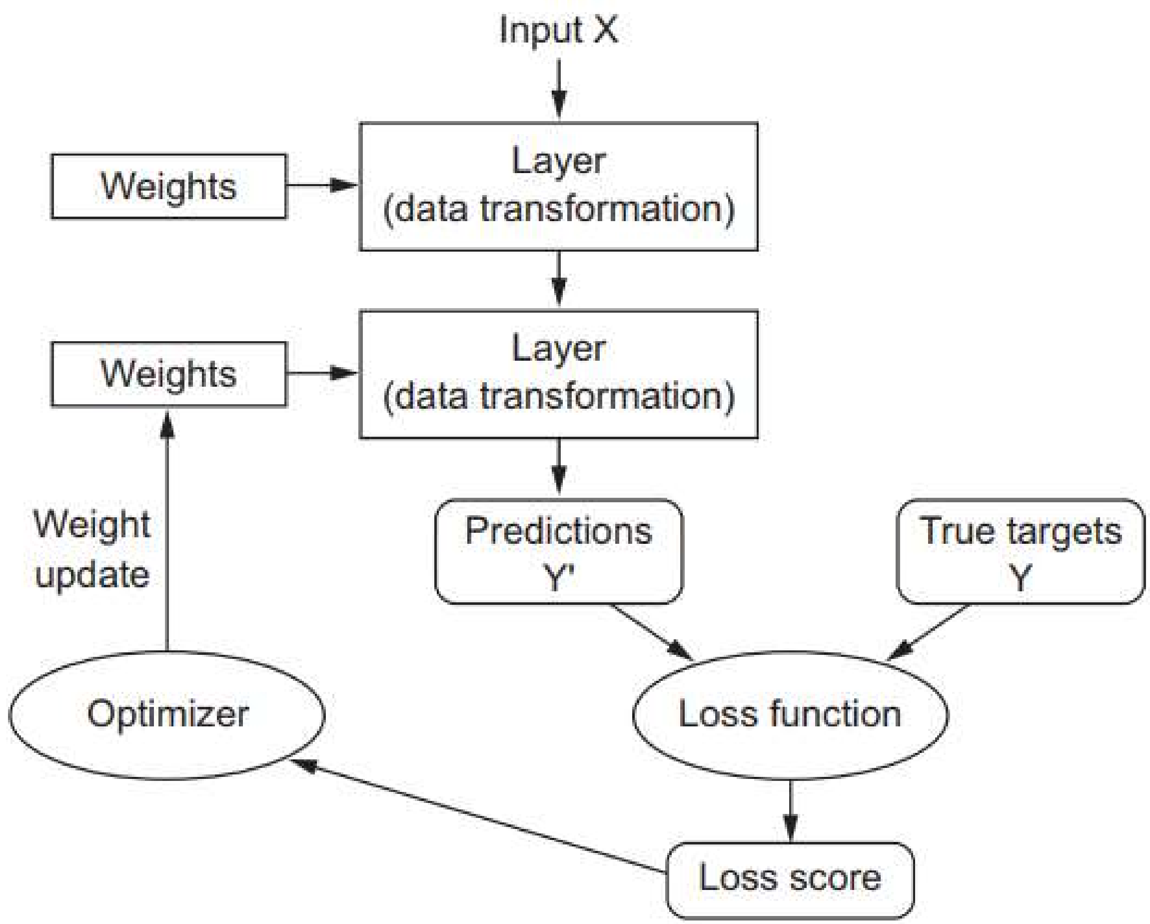
- Training a neural network revolves around the following objects:
  - *Layers*, which are combined into a *network* (or *model*)
  - The *input data* and corresponding *targets*
  - The *loss function*, which defines the feedback signal used for learning
  - The *optimizer*, which determines how learning proceeds

# Anatomy of neural networks

- Training a neural network revolves around the following objects:
  - *Layers*, which are combined into a *network* (or *model*)
  - The *input data* and corresponding *targets*
  - The *loss function*, which defines the feedback signal used for learning
  - The *optimizer*, which determines how learning proceeds

# Anatomy of neural networks

- Relationship between the network, layers, loss function, and optimizer



# Anatomy of neural networks

1. The building blocks of deep learning:
   - The fundamental data structure in neural networks is the layer.
   - A layer is a data processing module that takes as input one or more tensors and that outputs one or more tensors.
   - Different types of layers are appropriate for different tensor formats and different types of data processing

# Anatomy of neural networks

1. The building blocks of deep learning:
   - For instance:
     - Simple vector data:
       - ✓ Stored in rank-2 tensors of shape (samples, features),
       - ✓ Often processed by *densely connected* layers, also called *fully connected* or *dense* layers (the Dense class in Keras).
     - Sequence data:
       - ✓ Stored in rank-3 tensors of shape (samples, timesteps, features),
       - ✓ These are typically processed by *recurrent* layers, such as an LSTM layer, or 1D convolution layers (Conv1D).

# Anatomy of neural networks

1. The building blocks of deep learning:
   - ❖ For instance (cont):
     - ➢ Image data:
       - ✓ Stored in rank-4 tensors,
       - ✓ Usually, 2D convolution layers (Conv2D) process this data.

# Anatomy of neural networks

1. The building blocks of deep learning:
   - ❖ For instance (cont):
     - ➢ Image data:
       - ✓ Stored in rank-4 tensors,
       - ✓ Usually, 2D convolution layers (Conv2D) process this data.

# Anatomy of neural networks

2. Network of layers:
   - A deep-learning model is a directed, acyclic graph of layers.
   - The most common instance is a linear stack of layers, mapping a single input to a single output.
   - Some common ones include the following:
     - ✓ Two-branch networks
     - ✓ Multihead networks
     - ✓ Inception blocks

# Anatomy of neural networks

2. Network of layers:
   - The topology of a network defines a hypothesis space.
   - A network topology specifies a series of tensor operations, mapping input data to output data.
   - The process the network works on is to find a good set of values for the weight tensors.
   - Picking the right network architecture is more an art than a science.
   - Practice and practice can help you become a proper neural-network architect.

# Anatomy of neural networks

3. Loss functions and optimizers:
   - Once the network architecture has been defined, there are two more things to do:
     - *Loss function* (objective function)—The quantity that will be minimized during training. It represents a measure of success for the task at hand.
     - *Optimizer—Determines* how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent (SGD).

# Anatomy of neural networks

3. Loss functions and optimizers:
   - Neural networks with multiple outputs may have multiple loss functions.
   - The loss functions must be combined into a single scalar loss for gradient descent.
   - Choosing the right objective function is crucial because the network will optimize for it ruthlessly.
   - Potentially leading to unintended consequences if the function doesn't fully align with the desired outcome.
   - A poorly chosen objective can lead to harm.

# Anatomy of neural networks

3. Loss functions and optimizers:
   - For common tasks like classification, regression, and sequence prediction, there are standard loss functions:
     - ✓ Binary cross-entropy for two-class classification
     - ✓ Mean squared error for regression.
   - Custom loss functions are mainly needed for novel research problems.

# Introduction to Keras

- Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model.

- Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model.
- Keras was initially developed for researchers, with the aim of enabling fast experimentation.

# Introduction to Keras

- Keras is a high-level API for Neural Networks, written in Python that runs on TensorFlow
- **Goal**: go from idea to result with the least possible delay.
- TensorFlow is an open source software library for numerical computation using data flow graphs.

# Introduction to Keras

- Keras has the following key features:
  - ✓ It allows the same code to run seamlessly on a CPU or GPU.
  - ✓ It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
  - ✓ It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.

# Introduction to Keras

- Keras is distributed under the permissive MIT license => it can be freely used in commercial projects.
- Keras is compatible with any version of Python from 2.7 to 3.6 (as of mid-2017).
- Keras has well over 200,000 users.
- Keras is used at Google, Netflix, Uber, CERN, Yelp, Square, and hundreds of startups working on a wide range of problems.
- Keras is also a popular framework on Kaggle.

# Introduction to Keras

1. Keras, Tensorflow, Theano, CNTK:
   - Keras is a high-level library for building deep-learning models, relying on specialized tensor libraries for low-level operations.
   - Keras supports multiple backend engines, including TensorFlow, Theano, and CNTK, with potential for future expansion.

# Introduction to Keras

1. Keras, Tensorflow, Theano, CNTK:
   - Keras handles the problem in a modular way; thus, several different backend engines can be plugged seamlessly into Keras.
   - There are three existing backend implementations are the TensorFlow backend, the Theano backend, and the Microsoft Cognitive Toolkit (CNTK) backend.


The deep-learning software and hardware stack

# Introduction to Keras

1. Keras, Tensorflow, Theano, CNTK:
   - TensorFlow, CNTK, and Theano are major deep-learning platforms, developed by Google, Microsoft, and Université de Montréal, respectively.
   - Keras allows seamless switching between these backends without code changes, making development more flexible.
   - TensorFlow is recommended as the default due to its scalability and widespread use.
   - When running on CPU, TensorFlow is itself wrapping a low-level library for tensor operations called Eigen
   - Keras can run on both CPUs and GPUs, leveraging Eigen for CPU operations and NVIDIA's cuDNN for optimized GPU performance.

# Introduction to Keras

1. Keras, Tensorflow, Theano, CNTK:
   - Links:
     - ✓ Theano
       http://deeplearning.net/software/theano
     - ✓ TensorFlow
       www.tensorflow.org
     - ✓ CNTK
       https://github.com/Microsoft/CNTK
     - ✓ Eigen
       http://eigen.tuxfamily.org

# Introduction to Keras

2. Quick overview of Keras:
   - The typical Keras workflow looks just like that example:
     1. Define your training data: input tensors and target tensors.
     2. Define a network of layers (or model) that maps your inputs to your targets.
     3. Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.
     4. Iterate on your training data by calling the fit() method of your model.

# Introduction to Keras

2. Quick overview of Keras:
   - Keras supports two implementation models:
     1. Sequential
        - ✓ A set of linear layers in the form of a stack.
        - ✓ It requires a clear definition of the input data type for it.
        - ✓ Easy to learn
     2. Functional
        - ✓ A model that uses API functions
        - ✓ Similar to TensorFlow 2.0

- Reference: https://keras.io.

# Sequential() model

- Sequential model and deep learning: same as TensorFlow
  - ✓ Sequential models are suitable for a stack of simple layers where each layer has exactly one input tensor and one output tensor.
  - ✓ Steps to follow
    - Create model
    - Compile model (model.compile())
    - Fit model (model.fit())
    - Model-based prediction (model.predict())
    - Model Evaluation (model.evaluate())
    - Save model (model.save())
    - Load model (model.load())

# Sequential() model

- ❖ Sequential model and deep learning:
  - ✓ Declare the library packages to use

```
import tensorflow as tf
import keras
from keras import layers
```

# Sequential() model

❖ Sequential model and deep learning:

➢ Create model

✓ Define sequential model have 3 layer

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Gọi model với một dữ liệu đầu vào
x = tf.ones((3, 3))
y = model(x)
```

# Sequential() model

- ❖ Sequential model and deep learning:
  - ➢ Create model
  - ✓ Another way to declare

```
layer1 = layers.Dense(2, activation="relu", name="layer1")
layer2 = layers.Dense(3, activation="relu", name="layer2")
layer3 = layers.Dense(4, name="layer3")

x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

# Sequential() model

- Sequential model and deep learning:
  - Create model
  - ✓ Result of executing model creation using method summary()

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (3, 2)                    8


 layer2 (Dense)              (3, 3)                    9


 layer3 (Dense)              (3, 4)                    16


=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

# Sequential() model

- Sequential model and deep learning:
  - ✓ Compile model (model.compile())

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

# Sequential() model

- Sequential model and deep learning:
  - ✓ Fit model (model.fit())

```
model.fit(X_train, y_train, batch_size=batch_size,
          epochs=epoch, verbose=1,
          validation_data=(X_test, y_test),
          callbacks=[model_checkpoint_callback])
```

# Sequential() model

- Sequential model and deep learning:
  - ✓ Model-based prediction (model.predict())

```
pickle.dump(model, open(file_name, 'wb'))


# Dự đoán lớp trên dữ liệu test
predicted = model.predict(X_test)
```

# Sequential() model

❖ Sequential model and deep learning:

✓ Model Evaluation (model.evaluate())

```
scores = model.evaluate(X_test, y_test, verbose=1)
print("Loss:", (scores[0]))
print("Accuracy:", (scores[1]*100))
```

# Sequential() model

- Sequential model and deep learning
  - Save model (model.save())
    - ✓ Save the model to use in the next call without retraining

```
model.save_weights(model_h5_file)
```

  - Load model (model.load())
    - ✓ Read pre-trained model to use

```
model_temp = model.load_weights(model_h5_file)
```

# Keras functional APIs

- Keras APIs are used to create more flexible models than keras.Sequential API.
- APIs can handle models with non-linear topologies, shared layers, and even multiple inputs or outputs.
- The main idea behind a deep learning model is a directed acyclic graph (DAG) of layers. So APIs are a way to build a graph of layers.

# Keras functional APIs

- Example:
  a. Building the model: defining inputs and outputs

```
inputs = keras.Input(shape=(784,))
img_inputs = keras.Input(shape=(32, 32, 3))
dense = layers.Dense(64, activation="relu")
x = dense(inputs)
x = layers.Dense(64, activation="relu")(x)
outputs = layers.Dense(10)(x)
model = keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
model.summary()
```

# Keras functional APIs

- Example:
  - Results of modeling

```
Model: "mnist_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 55,050
Trainable params: 55,050
Non-trainable params: 0
```

# Keras functional APIs

- Example:

  b. Training, Evaluation, and Inference

  - ✓ The training, evaluation, and inference processes work similarly to Sequential.
  - ✓ The **Model** class provides a training loop using the fit() method.
  - ✓ During training, the **Model** performs validation.
  - ✓ The process of loading MNIST image data involves reshaping the data into vectors, fitting the model to the data.
  - ✓ Evaluate the performance of the **Model** on test data using the evaluate() method.

# Keras functional APIs

❖ Example:

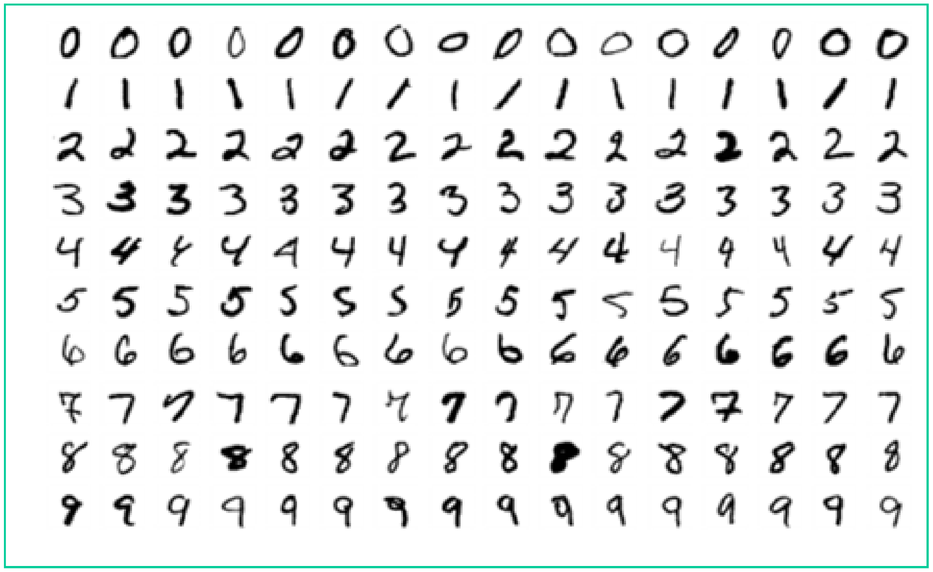
b. Training, Evaluation, and Inference

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape(60000, 784).astype("float32") / 255
x_test = x_test.reshape(10000, 784).astype("float32") / 255

model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.RMSprop(),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

history = model.fit(x_train, y_train, batch_size=64, epochs=2, validation_split=0.2)
test_scores = model.evaluate(x_test, y_test, verbose=2)
print("Test loss:", test_scores[0])
print("Test accuracy:", test_scores[1])
```

# Keras functional APIs

- Example:
  b. Training, evaluation, and inference

  - The corpus used in the example is MINST (Modified National Institute of Standards and Technology).
  - MINST contains handwritten digits that are commonly used in training image processing systems.



# Keras functional APIs

❖ Example:

b. Training, Evaluation and Inference

➢ Implementation results:

```
Epoch 1/2
750/750 [==============================] - 3s 3ms/step - loss: 0.3431 - sparse_ca
Epoch 2/2
750/750 [==============================] - 2s 3ms/step - loss: 0.1582 - sparse_ca
313/313 - 0s - loss: 0.1514 - sparse_categorical_accuracy: 0.9565 - 298ms/epoch -
Test loss: 0.15140439569950104
Test accuracy: 0.9564999938011169
```

# Keras functional APIs

- Example:
  c. Save model
  - Saving the model is similar to Sequential.
  - Call the save() method to save the entire model as a file.
  - Saving the model as a file allows the same model to be recreated from this file, even if the model generation code is no longer available.
  - This saved file contains:
    - ✓ The model architecture
    - ✓ The model weights (obtained from training)
    - ✓ The model training configuration, if any (passed to compilation)
    - ✓ The optimizer and its state (to restart training where it left off)

# Keras functional APIs

❖ Example:

c. Save model

```
model.save("path_to_my_model.keras")
del model


model = keras.models.load_model("path_to_my_model.keras")
```