Głębokie modele uczenia maszynowego

Środa 9.15-10.45

Kamil Tagowski (kamil.tagowski@pwr.edu.pl)

Plan zajęc

Zajęcia laboratoryjne składają się z dwóch części:

- Część I: Laboratoryjna (40 pkt)
- Cześć II: Laboratoryjna lub projektowa (60 pkt)

Ocenianie

Punktacja	0-50	51-60	61-70	71-80	81-90	91-100
Ocena	2	3	3.5	4	4.5	5

Część pierwsza

	Daty	Termin oddania	Maksymalna Punktacja
Wprowadzenie do zajęć laboratoryjnych	02.10.2019	-	-
Ćw. 1: MLP - Implementacja architektury	09.10.2019	14.10.2019	5 pkt
Ćw. 1: MLP - Badanie hiperparametrów	16.10.2019	21.10.2019	5 pkt
Ćw. 2: CNN - Implementacja architektury	23.10.2019	28.10.2019	5 pkt
Ćw. 2: CNN - Badanie połączeń rezudualnych	30.10.2019	04.11.2019	5 pkt
Ćw. 3: RNN - Implementacja architektury	06.11.2019	18.11.2019	5 pkt
Ćw. 3: RNN - Badanie reprezentacji wektorowej słów	20.11.2019	25.11.2019	5 pkt
Konkurs	15.11.2019-15.12.2019	-	10 pkt

Część druga

Ścieżka projektowa

	Daty	Termin oddania	Maksymalna Punktacja
Zajęcia wprowadzające	27.11.2019	-	-
Przegląd literaturowy	04.12.2019, 11.12.2019	16.12.2019	12 pkt
Implementacja	18.12.2019, 08.01.2020	13.01.2020	18 pkt
Badania + raport	15.01.2020, 22.01.2020	27.01.2020	30 pkt

Ścieżka laboratoryjna

	Daty	Termin oddania	Maksymalna Punktacja
Zajęcia wprowadzające	27.11.2019	-	-
Ćw 4.: Implementacja	04.12.2019, 11.12.2019	15.12.2019	18 pkt
Ćw 4.: Poszukiwanie hiperparametrów	18.12.2019	05.01.2020	12 pkt
Ćw 5.: Implementacja	08.01.2020, 15.01.2020	20.01.2020	18 pkt
Ćw 5.: Poszukiwanie hiperparametrów	22.01.2020	27.01.2020	12 pkt

Regulamin

- Za opóźnienie o każdy termin zajęć kara punktowa wynosi 20% punktacji bazowej;
- Oddanie zadania w terminie późniejszym niż 3 tygodnie od zaplanowanego terminu skutkuje otrzymaniem 0 punktów za to zadanie;
- Aktywny udział w zajęciach laboratoryjnych jest obowiązkowy (liczy się praca w trakcie zajęć oraz w domu);

Wstęp do Tensorflow 2.0

Biblioteka do definiowania grafów obliczeniowych

- **Oficjalne repozytorium**
  https://github.com/tensorflow/tensorflow (https://github.com/tensorflow/tensorflow)
- **Dokumentacja i tutoriale**
  https://www.tensorflow.org/tutorials/ (https://www.tensorflow.org/tutorials/)

Instalacja Tensorflow

CPU

```
pip install tensorflow==2.0.0-rc2
```

CPU/GPU

```
pip install tensorflow-gpu==2.0.0-rc2
```

Więcej informacji lub instalacja za pomocą Dockera
https://www.tensorflow.org/install/ (https://www.tensorflow.org/install/)

Tryb eager vs statyczny graf obliczeniowy

Statyczny graf obliczeniowy (Tensorflow 1.x)

```
import tensorflow as tf

m = tf.placeholder(tf.float32)
op = tf.matmul(m, m)

with tf.Session() as sess:
    y = sess.run(op, feed_dict={a: [3]})

print('Result: ', y)
```

Tryb eager (Tensorflow 2.x)

Pierwsza wersja wprowadzona w TensorFlow 1.7

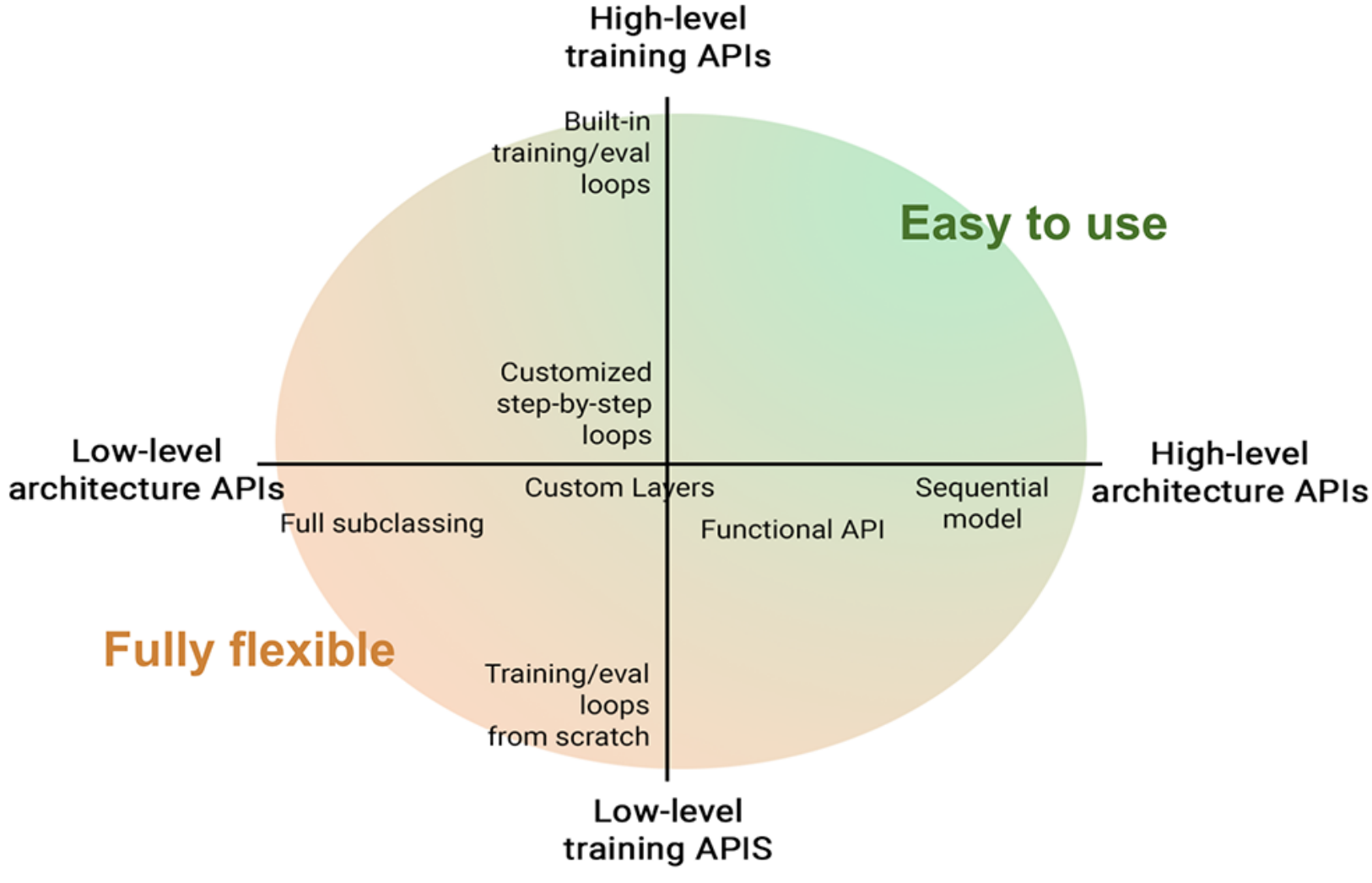
```
import tensorflow as tf
tf.compat.v1.enable_eager_execution()

x = [[3.]]
print(tf.matmul(x,x).numpy())
```

Dostępne interfejsy programistyczne:

- Funkcyjny **(Functional API)**
- Sekwencyjny **(Sequential API)**

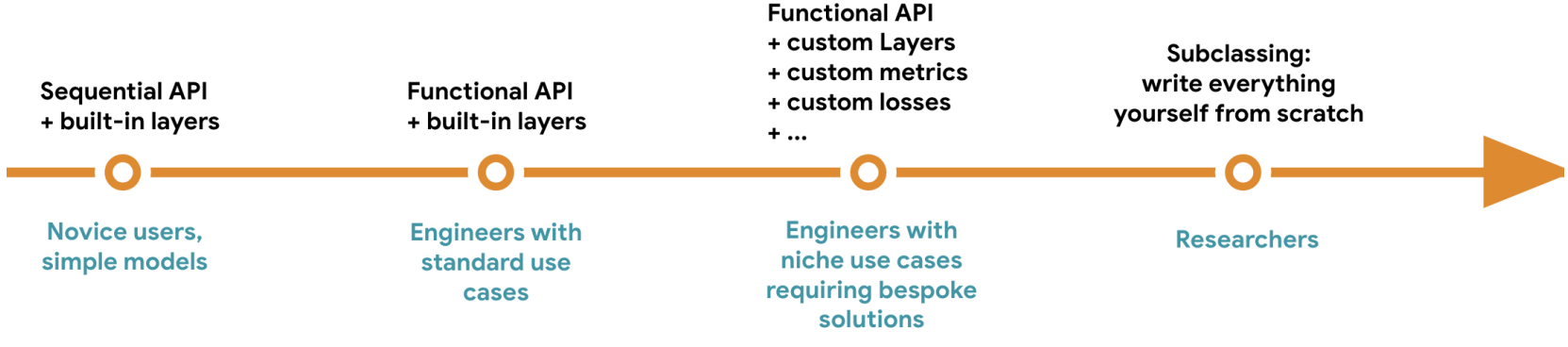
Interejs na którym będą oparte zadania

- Klasowy **(Subclassing API)**


High-level
training APIs
Built-in
training/eval
loops
Easy to use
Customized
step-by-step
loops
Low-level
architecture APIs
High-level
architecture APIs
Custom Layers
Sequential
model
Full subclassing
Functional API
Fully flexible
Training/eval
loops
from scratch
Low-level
training APIS

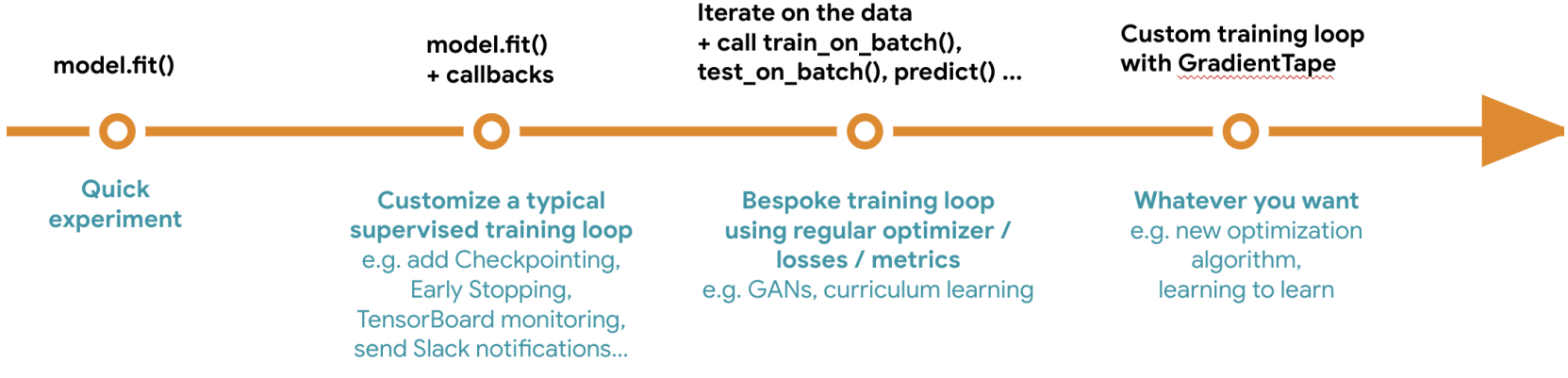
Model building: from **simple** to **arbitrarily flexible**

Progressive disclosure of complexity



Model training: from **simple** to **arbitrarily flexible**

Progressive disclosure of complexity



Functional API

```
# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
output_1 = Dense(64, activation='relu')(inputs)
output_2 = Dense(64, activation='relu')(output_1)
predictions = Dense(10, activation='softmax')(output_2)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Sequential API

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Subclassing API

```
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.flatten = Flatten()
    self.d1 = Dense(64, activation='relu')
    self.d2 = Dense(64, activation='relu')
    self.d3 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.flatten(x)
    x = self.d1(x)
    x = self.d2(x)
    return self.d3(x)


model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Popularne narzędzia do wizualizacji

- TensorBoard (TensorFlow, TensorBoardX - PyTorch)
- Visdom (PyTorch, Numpy)