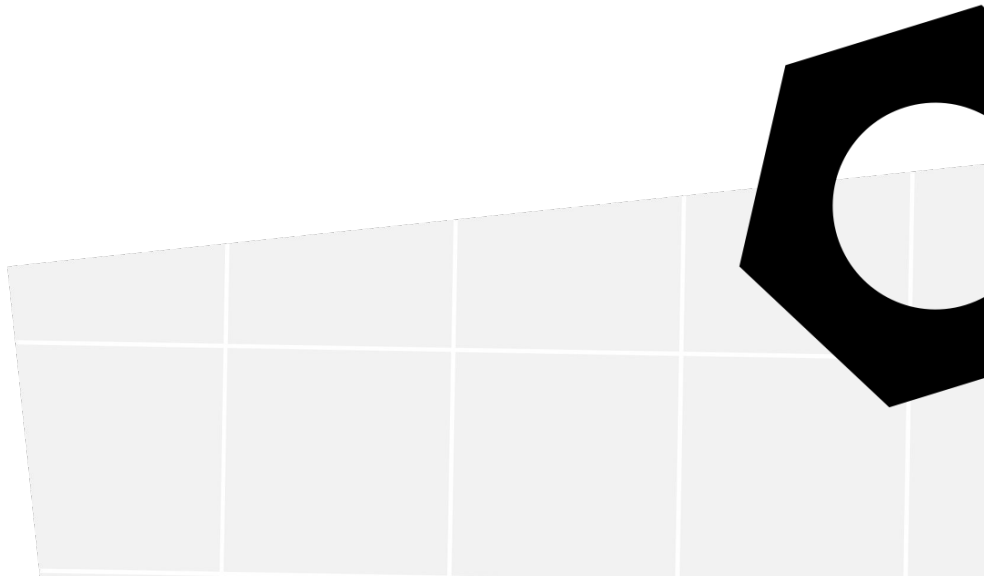
Biblioteka TensorFlow

Kurs Data Science



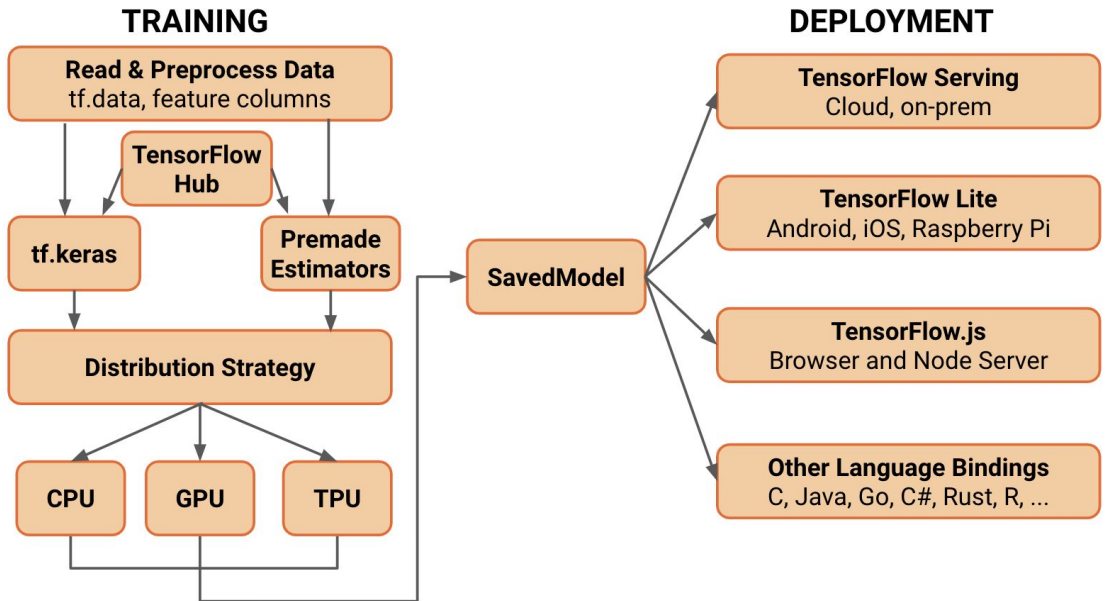
software
development
academy

Do pobrania

Podczas zajęć będziemy korzystać z dodatkowego pliku - możesz go pobrać z sekcji “Dodatkowe materiały do bloku” tutaj.

Wprowadzenie do tensorflow

- **TensorFlow** został zaprojektowany przez naukowców i inżynierów Google, aby realizować projekty badawcze w dziedzinie maszynowego uczenia się (machine learning) oraz głębokich sieci neuronowych (deep neural networks). System pozostaje mimo to dość ogólny, co pozwala na szeroką gamę innych zastosowań. TensorFlow umożliwia przede wszystkim optymalne uczenie się modeli wymagających dużej ilości danych (na przykład banków zdjęć).

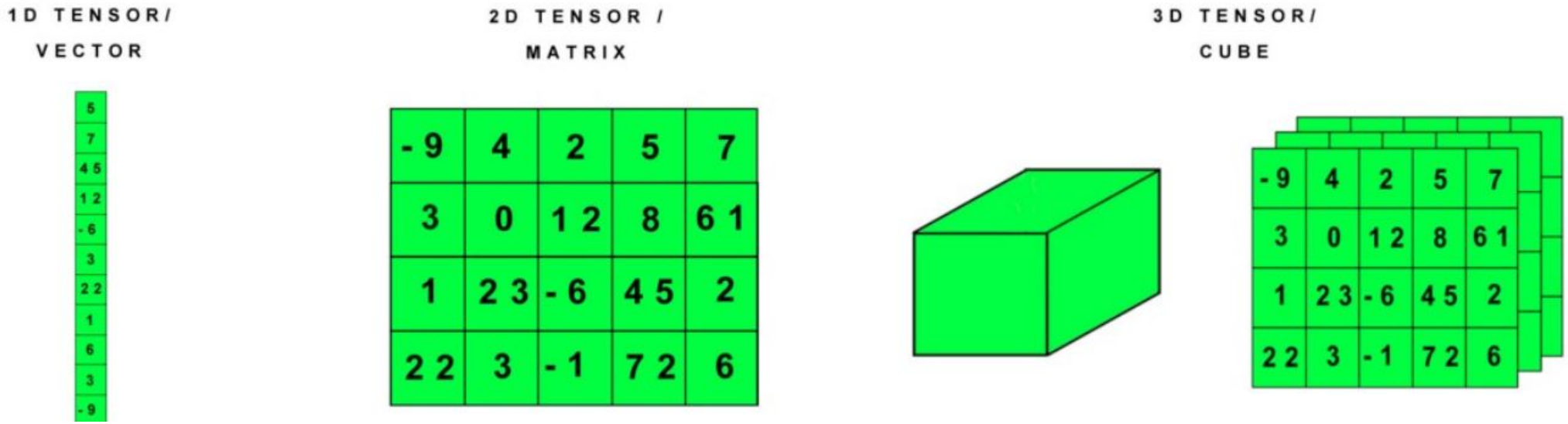


Wprowadzenie do tensorflow

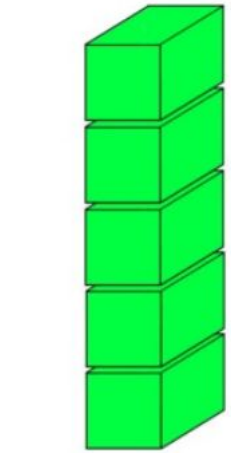
- Elastyczna architektura TensorFlow umożliwia wdrożenie obliczeń na jednym lub większej liczbie procesorów (CPU) lub kart graficznych (GPU) na komputerze osobistym, serwerze *itp.* **bez konieczności ponownego pisania kodu**.
- Firma Google stworzyła także **Tensor Processing Units** (inaczej **TPU**), wyspecjalizowany układ scalony, zaprojektowany z myślą o rozwoju maszynowego uczenia się i stosowania TensorFlow. TPU zostały zaprojektowane do wykorzystywania i testowania modeli, a nie do ich uczenia się. Od lutego 2018 r. TPU są dostępne w wersji beta **Google Cloud Platform**.

Czym jest tensor?

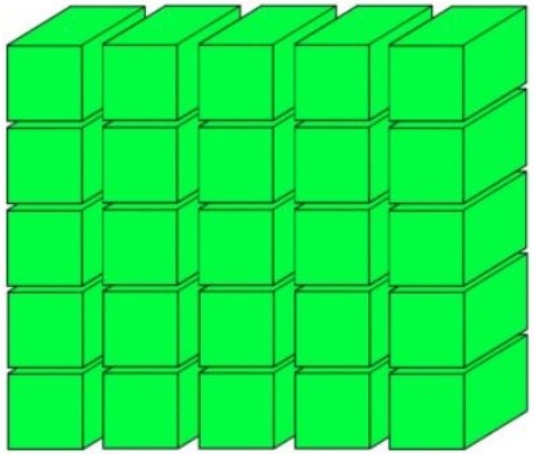
Czasem potrzebujemy tablic o większej liczbie wymiarów. Ogólnie rzecz biorąc liczby tablicy tworzą regularną siatkę ze zmienną liczbą osi, którą nazywamy tensorem. Tensor jest "pojemnikiem" na dane. Możemy mieć tensor 0D, do którego mieści się jeden skalar. Tensor 1D jest nazywany wektorem. Tensor 2D to macierz.



Czym jest tensor?



4D TENSOR
VECTOR OF CUBES



5D TENSOR
MATRIX OF CUBES

Często potrzebujemy przechowywać wiele tensorów 2D razem. Otrzymujemy wtedy tensor 3D, czyli sześcian liczb. Możemy łączyć owe kostki / sześciany ze sobą, tworząc tensory 4D, 5D,..., ND.

Tensory oznaczamy przy pomocy pogrubionej czcionki i drukowanych liter. Przykładowo element tensora $\mathbf{A}$ o współrzędnych (i,j,k) zapisujemy następująco: $\mathbf{A}_{i,j,k}$. W uczeniu maszynowym tensory wykorzystujemy do przechowywania różnych typów danych, np. tensory 3D do szeregów czasowych, 4D do obrazów, a 5D do wideo.

Rozpoczęcie pracy z tensorflow

Szczegóły instalacji mogą zależeć od posiadanego sprzętu, zaleca się korzystanie z oficjalnej dokumentacji tf (https://www.tensorflow.org/api_docs/python/tf).

Będziemy używać api tensorflow dla Pythona, dlatego zalecamy zainstalowanie również programu Anaconda.

Tensorflow API

API TensorFlow są podzielone na moduły:

https://www.tensorflow.org/api_docs/python/tf

Modules

`audio` module: Public API for tf.audio namespace.

`autodiff` module: Public API for tf.autodiff namespace.

`autograph` module: Conversion of eager-style Python into TensorFlow graph code.

`bitwise` module: Operations for manipulating the binary representations of integers.

`compat` module: Compatibility functions.

`config` module: Public API for tf.config namespace.

`data` module: `tf.data.Dataset` API for input pipelines.

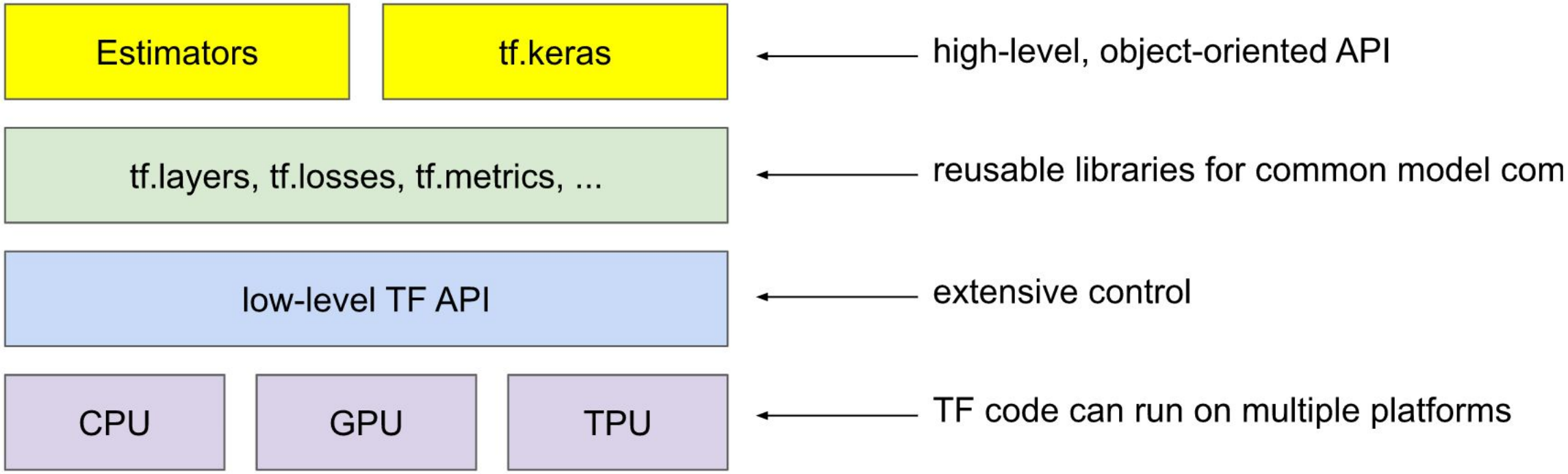
`debugging` module: Public API for tf.debugging namespace.

`distribute` module: Library for running a computation across multiple devices.

`dtypes` module: Public API for tf.dtypes namespace.

`errors` module: Exception types for TensorFlow errors.

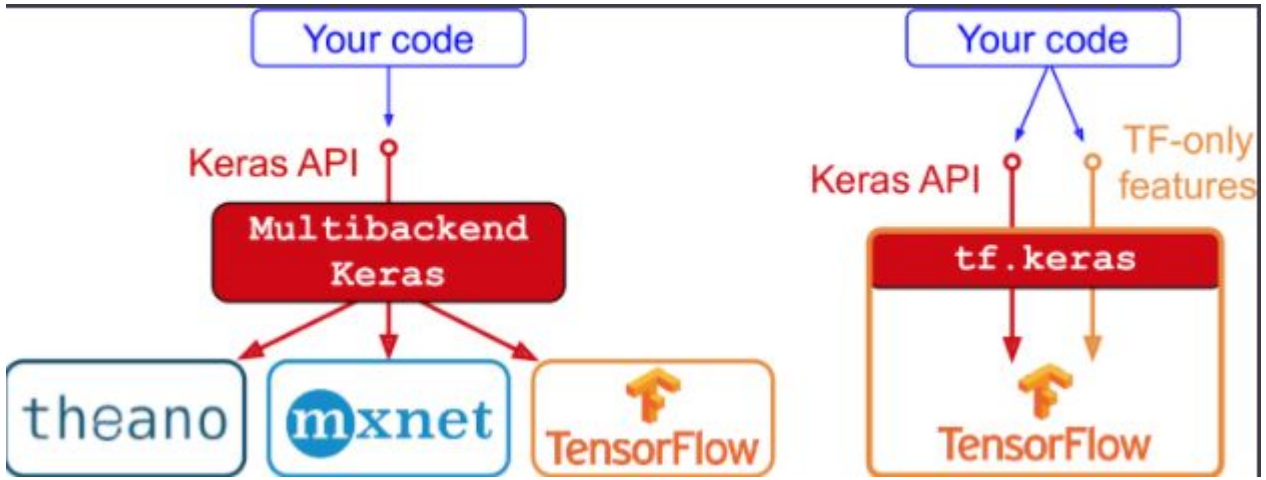
Tensorflow API



Keras and tf.keras

keres.io - implementacja referencyjna dla Tensorflow

tf.keras - implementacja tensorflow (podzbiór pakietu Tensorflow)



Zbiory Danych TensorFlow

- dźwięk
  - nsynth
- zdjęcie
  - cifar10
  - diabetic_retionpathy_detection
  - imagenet2012
  - mnist
- struktura
  - titanic
- tekst
  - imdb_reviews
  - lm1b
  - squad
- tłumaczenie
  - wmt_translate_ende
  - wmt_translate_enfr
- video
  - bair_robot_pushing_small
  - moving_mnist
  - starcraft_video

Więcej: tensorflow.org/datasets

Typy danych w tensorflow api

Typy danych API Pythona w TensorFlow	Opis
`tf.float16`	16-bitowa liczba zmiennoprzecinkowa (połowiczna precyzja)
`tf.float32`	32-bitowa liczba zmiennoprzecinkowa (pojedyncza precyzja)
`tf.float64`	64-bitowa liczba zmiennoprzecinkowa (podwójna precyzja)
`tf.int8`	8-bitowa liczba całkowita (ze znakiem)
`tf.int16`	16-bitowa liczba całkowita (ze znakiem)
`tf.int32`	32-bitowa liczba całkowita (ze znakiem)
`tf.int64`	64-bitowa liczba całkowita (ze znakiem)

Definiując tensory używaj typu danych z tensorflow a nie np. z numpy.

tf.Tensor

Reprezentuje wielowymiarową tablicę elementów, wszystkie elementy są jednego znanego typu danych. tf.Tensor jest niezmienny.

Właściwości:

pojedynczy typ danych

- kształt

W wykonaniu eager operacje są obliczane natychmiast

Dostępne są specjalistyczne tensory:

- tf.Variable
- tf.constant
- tf.sparse.SparseTensor
- tf.RaggedTensor

tensorflow.python.framework.ops - kod źródłowy

Inicjalizacja stałych tensorów

- Do tej pory dyskutowaliśmy o tensorach jako abstrakcyjnych bytach matematycznych. Jednak system taki jak TensorFlow musi działać na prawdziwym komputerze, więc każdy tensor musi funkcjonować w pamięci komputera, aby być użyteczny dla programistów. TensorFlow zapewnia szereg funkcji, które inicjują podstawowe tensory w pamięci. Najprostsze z nich to tf.zeros() i tf.ones(). tf.zeros() przybiera kształt tensora (reprezentowanego jako krotka Pythona) i zwraca tensor o tym kształcie wypełniony zerami.
- Utworzenie tensora z zerowymi elementami:

```
tf.zeros(2)
```

```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([0., 0.], dtype=float32)>
```

Inicjalizacja stałych tensorów

- TensorFlow zwraca odwołanie do żądanego tensora, a nie jego wartość. Aby wymusić zwrócenie wartości tensora, użyjemy metody tf.Tensor.eval() obiektów tensora. Ponieważ zainicjowaliśmy tf.InteractiveSession(), metoda ta zwróci nam wartość zer tensora.
- Obliczanie wartości tensora

```
a = tf.zeros(2)

a.eval()
```

Inicjalizacja stałych

```
import tensorflow as tf

x = tf.constant([[1., 2., 3.],
                 [4., 5., 6.]])

print(x)
print(x.shape)
print(x.dtype)
```

```
tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
(2, 3)
<dtype: 'float32'>
```

Broadcasting

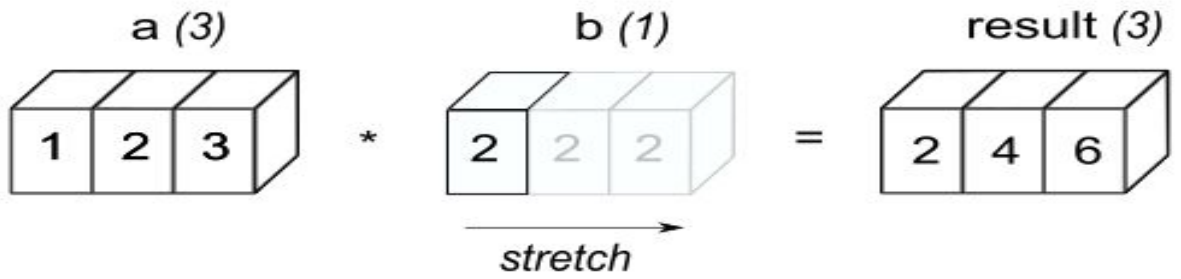
Koncepcja ta została zapożyczona z analogicznej funkcji w NumPy.

```
a = tf.constant([1.0, 2.0, 3.0])
b = tf.constant([2.0, 2.0, 2.0])
a * b
```

```
<tf.Tensor: shape=(3,), dtype=float32, numpy=array([2., 4., 6.], dtype=float32)>
```

```
a = tf.constant([1.0, 2.0, 3.0])
b = 2.0
a * b
```

```
<tf.Tensor: shape=(3,), dtype=float32, numpy=array([2., 4., 6.], dtype=float32)>
```



```
x + x
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[ 2.,  4.,  6.],
       [ 8., 10., 12.]], dtype=float32)>
```

```
5 * x
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[ 5., 10., 15.],
       [20., 25., 30.]], dtype=float32)>
```

```
x @ tf.transpose(x)
```

```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[14., 32.],
       [32., 77.]], dtype=float32)>
```

```
tf.concat([x, x, x], axis=0)
```

```
<tf.Tensor: shape=(6, 3), dtype=float32, numpy=
array([[1., 2., 3.],
       [4., 5., 6.],
       [1., 2., 3.],
       [4., 5., 6.],
       [1., 2., 3.],
       [4., 5., 6.]], dtype=float32)>
```

```
tf.nn.softmax(x, axis=-1)
```

```
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[0.09 , 0.245, 0.665],
       [0.09 , 0.245, 0.665]], dtype=float32)>
```

tf.Variable

Obiekty tf.Tensor są niezmienne. Do przechowywania wag modeli lub innych zmiennych danych w TensorFlow możemy użyć tf.Variable

```
var = tf.Variable([0.0, 0.0, 0.0])
var.assign([1, 2, 3])
```

```
<tf.Variable 'UnreadVariable' shape=(3,) dtype=float32, numpy=array([1., 2., 3.], dtype=float
32)>
```

-

```
var.assign_add([1, 1, 1])
```

```
<tf.Variable 'UnreadVariable' shape=(3,) dtype=float32, numpy=array([2., 3., 4.], dtype=float3
2)>
```


tensorflow.python.framework.ops - soruce code

Automatyczne różnicowanie

Tensorflow implementuje automatyczne różnicowanie (autodiff), które wykorzystuje rachunek do obliczania gradientów.

$$f(x) = x^2 + 2x - 5$$

```
x = tf.Variable(1.0)

def f(x):
    y = x**2 + 2*x - 5
    return y
```

$$x = 1.0$$
$$y = f(x) = (1^2 + 2 * 1 - 5) = -2$$

```
with tf.GradientTape() as tape:
    y = f(x)
g_x = tape.gradient(y, x)  # g(x) = dy/dx
g_x
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=4.0>
```

Pętle treningowe

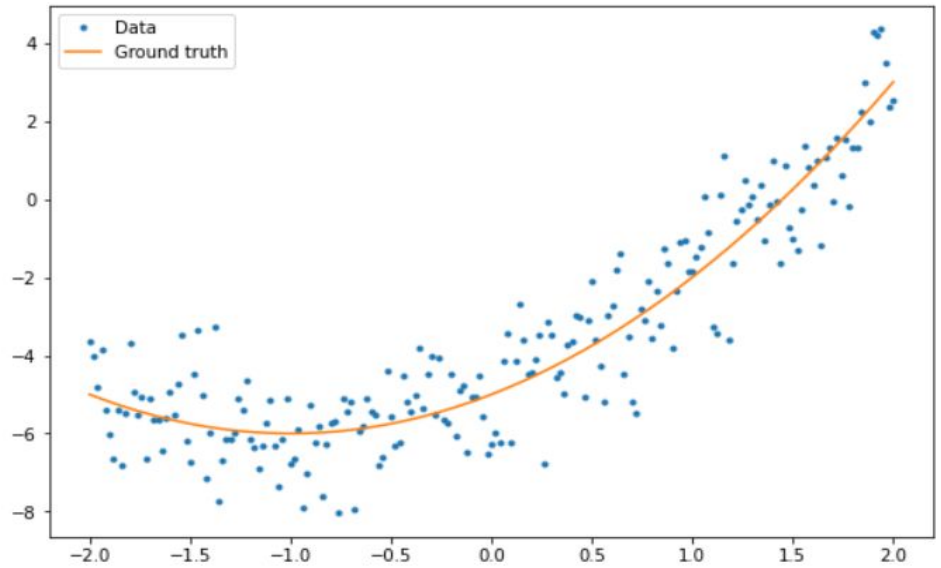
```
x = tf.linspace(-2, 2, 201)
x = tf.cast(x, tf.float32)

def f(x):
    y = x**2 + 2*x - 5
    return y

y = f(x) + tf.random.normal(shape=[201])

plt.plot(x.numpy(), y.numpy(), '.', label='Data')
plt.plot(x, f(x),  label='Ground truth')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa0aea55ee0>
```




```
model.summary()
```

```
Model: "model_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            multiple                  128

 dense_22 (Dense)            multiple                  65

=================================================================
Total params: 193
Trainable params: 193
Non-trainable params: 0
_________________________________________________________________
```

Tworzenie niestandardowego modelu liniowego

```python
# create custom model with base class tf.keras.Model
class Model(tf.keras.Model):
    def __init__(self, units):
        super().__init__()
        self.dense1 = tf.keras.layers.Dense(units=units,
                                            activation=tf.nn.relu,
                                            kernel_initializer=tf.random.normal,
                                            bias_initializer=tf.random.normal)
        self.dense2 = tf.keras.layers.Dense(1)

    def call(self, x, training=True):
        # For Keras layers/models, implement `call` instead of `__call__`.
        x = x[:, tf.newaxis]
        x = self.dense1(x)
        x = self.dense2(x)
        return tf.squeeze(x, axis=1)
```

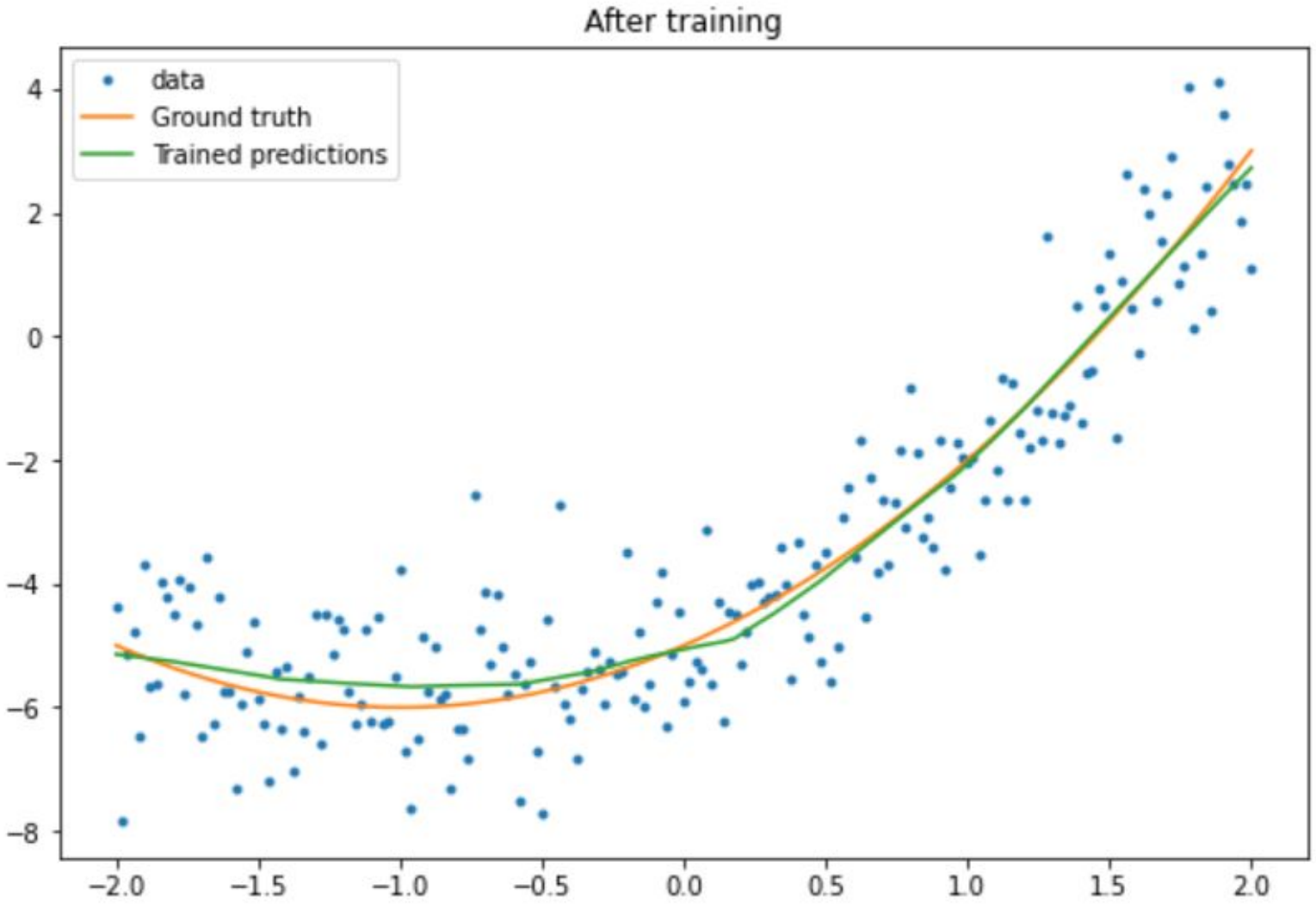
Podstawowa pętla treningowa

```python
# training loop
variables = model.variables
optimizer = tf.optimizers.SGD(learning_rate=0.01)
for step in range(1000):
    with tf.GradientTape() as tape:
        prediction = model(x)
        error = (y-prediction)**2
        mean_error = tf.reduce_mean(error)
    gradient = tape.gradient(mean_error, variables)
    optimizer.apply_gradients(zip(gradient, variables))

    if step % 100 == 0:
        print(f'Mean squared error: {mean_error.numpy():0.3f}')
```

```
Mean squared error: 38.687
Mean squared error: 1.072
Mean squared error: 1.063
Mean squared error: 1.056
Mean squared error: 1.050
Mean squared error: 1.044
Mean squared error: 1.038
Mean squared error: 1.035
Mean squared error: 1.032
Mean squared error: 1.030
```

```
plt.plot(x.numpy(),y.numpy(), '.', label="data")
plt.plot(x, f(x),  label='Ground truth')
plt.plot(x, model(x), label='Trained predictions')
plt.title('After training')
plt.legend();
```



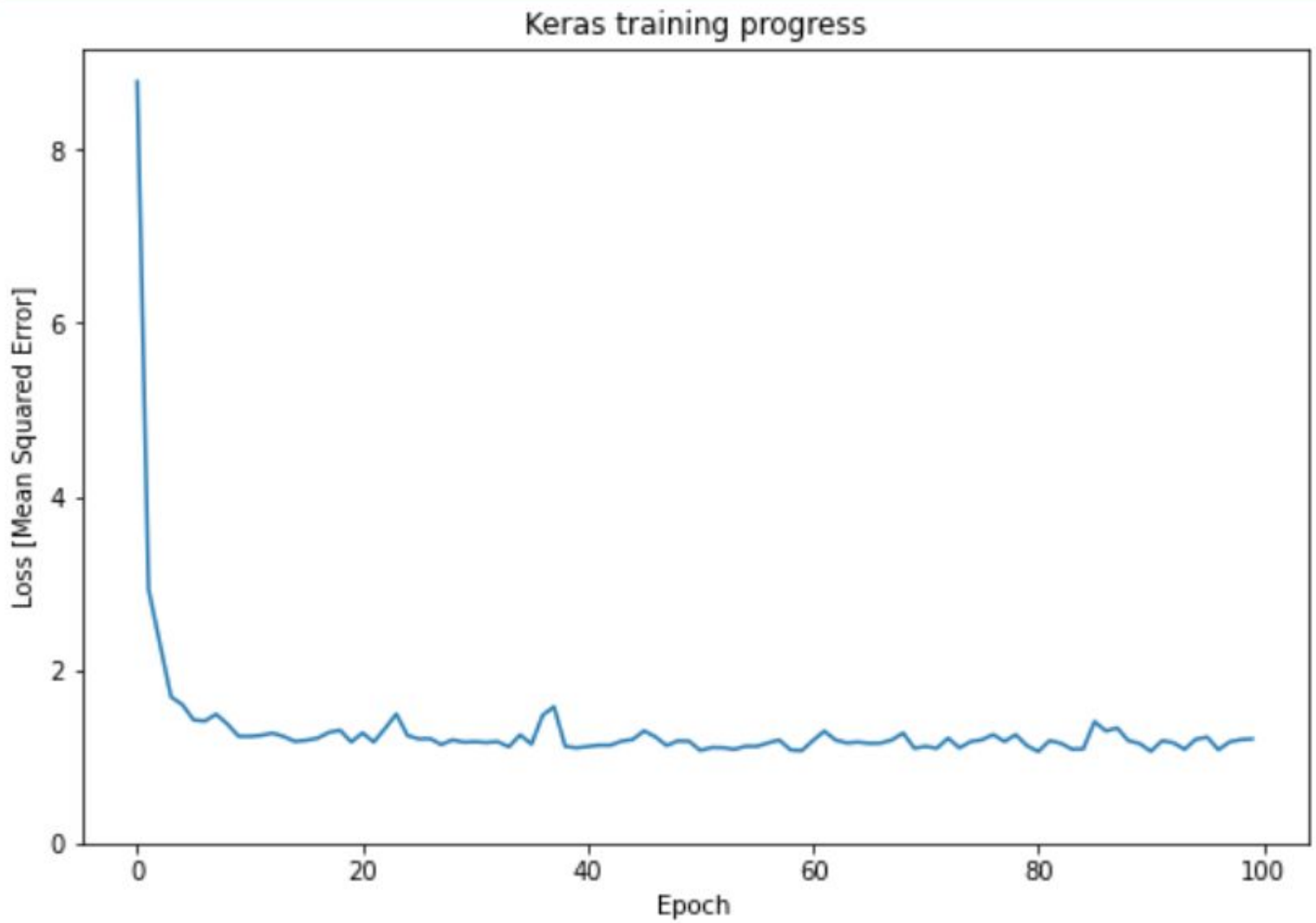
Z tf.keras

```
new_model = Model(64)
new_model.compile(
    loss=tf.keras.losses.MSE,
    optimizer=tf.optimizers.SGD(learning_rate=0.01))

history = new_model.fit(x, y,
                        epochs=100,
                        batch_size=32,
                        verbose=0)

model.save('./my_model')
plt.plot(history.history['loss'])
plt.xlabel('Epoch')
plt.ylim([0, max(plt.ylim())])
plt.ylabel('Loss [Mean Squared Error]')
plt.title('Keras training progress');
```

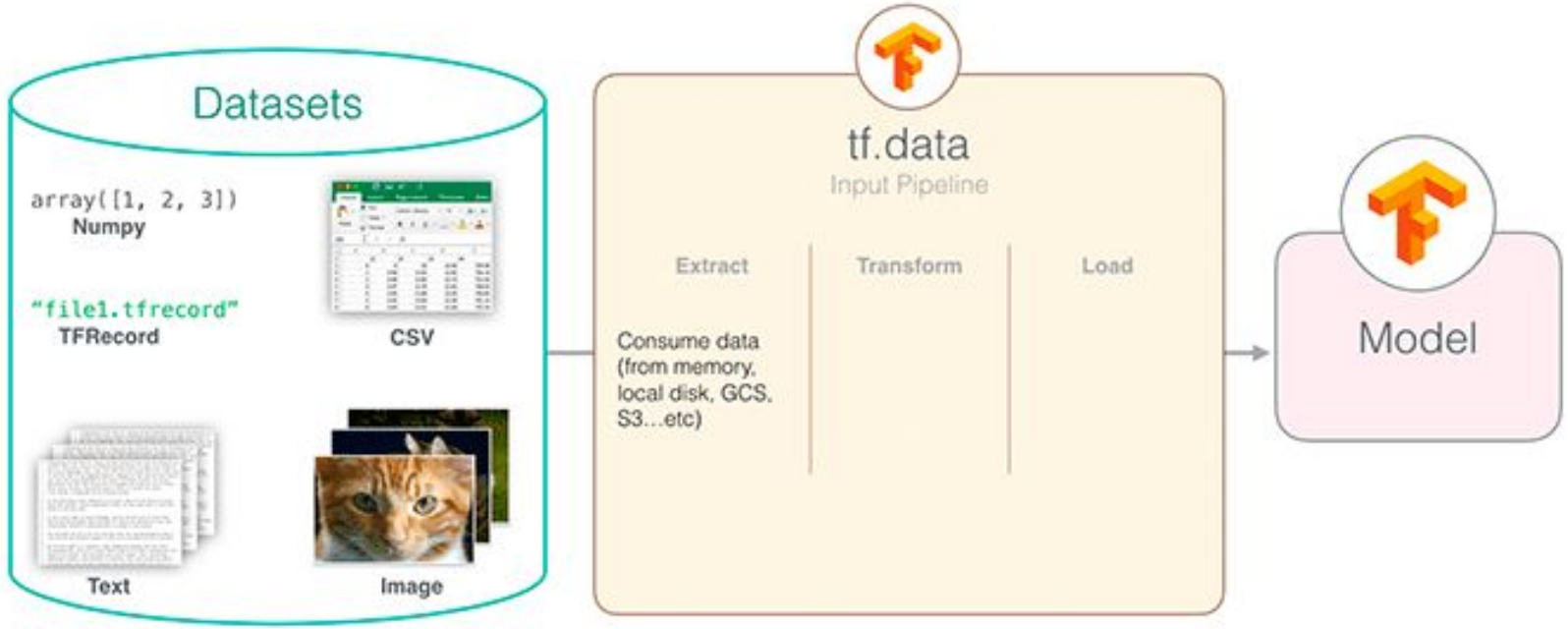
Z tf.keras


Keras training progress
Loss [Mean Squared Error]
Epoch
0
2
4
6
8
0
20
40
60
80
100

Z tf.keras

tf.data jest nowym API wprowadzonym w Tensorflow 2, umożliwiającym budowanie złożonych pipelinów wejściowych z prostych elementów.

tf.data.Dataset abstrakcja, która reprezentuje sekwencję elementów



tf.data - Odczyt danych wejściowych

tf.data może odczytywać dane wejściowe z różnych źródeł:

- Tablice NumPy
- Generatory Pythona
- TFRecord
- Dane tekstowe
- Dane CSV
- Zestawy plików

tf.data - Odczyt danych wejściowych

```
dataset = tf.data.Dataset.from_tensor_slices([8, 3, 0, 8, 2, 1])
dataset
```

```
<TensorSliceDataset element_spec=TensorSpec(shape=(), dtype=tf.int32, name=None)>
```

```
for elem in dataset:
    print(elem.numpy())
```

```
8
3
0
8
2
1
```

tf.data – tablice NumPy

```
train, test = tf.keras.datasets.fashion_mnist.load_data()
images, labels = train
images = images/255

dataset = tf.data.Dataset.from_tensor_slices((images, labels))
```

```
dataset
```

```
<TensorSliceDataset element_spec=(TensorSpec(shape=(28, 28), dtype=tf.float64, name=No
ne), TensorSpec(shape=(), dtype=tf.uint8, name=None))>
```

tf.data – dane tekstowe

```
directory_url = 'https://storage.googleapis.com/download.tensorflow.org/data/illiad/'
file_names = ['cowper.txt', 'derby.txt', 'butler.txt']

file_paths = [
    tf.keras.utils.get_file(file_name, directory_url + file_name)
    for file_name in file_names
]
```

```
dataset = tf.data.TextLineDataset(file_paths)
for line in dataset.take(5):
    print(line.numpy())
```

```
b"\xef\xbb\xbfAchilles sing, O Goddess! Peleus' son;"
b'His wrath pernicious, who ten thousand woes'
b"Caused to Achaia's host, sent many a soul"
b'Illustrious into Ades premature,'
b'And Heroes gave (so stood the will of Jove)'
```

tf.data – przetwarzanie danych

```
flowers_root = tf.keras.utils.get_file(
    'flower_photos',
    'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_phot
    untar=True)
flowers_root = pathlib.Path(flowers_root)
```

```
list_ds = tf.data.Dataset.list_files(str(flowers_root/'*/*'))
for f in list_ds.take(5):
    print(f.numpy())
```

```
def parse_image(filename):
    parts = tf.strings.split(filename, os.sep)
    label = parts[-2]
    image = tf.io.read_file(filename)
    image = tf.io.decode_jpeg(image)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, [128, 128])
    return image, label
```

```
file_path = next(iter(list_ds))
image, label = parse_image(file_path)
```

tf.data – przetwarzanie danych

```
def show(image, label):
    plt.figure()
    plt.imshow(image)
    plt.title(label.numpy().decode('utf-8'))
    plt.axis('off')

show(image, label)
images_ds = list_ds.map(parse_image)

for image, label in images_ds.take(4):
    show(image, label)
```

tf.data – time series windowing

```
range_ds = tf.data.Dataset.range(100000)
```

```
batches = range_ds.batch(10)
for batch in batches.take(5):
    print(batch.numpy())
```

```
[0 1 2 3 4 5 6 7 8 9]
[10 11 12 13 14 15 16 17 18 19]
[20 21 22 23 24 25 26 27 28 29]
[30 31 32 33 34 35 36 37 38 39]
[40 41 42 43 44 45 46 47 48 49]
```

```
def dense_1_step(batch):
  # shift features and labels one step left
  return batch[:-1], batch[1:]
predict_dense_1_step = batches.map(dense_1_step)

for features, label in predict_dense_1_step.take(3):
    print(features.numpy(), " => ", label.numpy())
```

```
[0 1 2 3 4 5 6 7 8]  =>  [1 2 3 4 5 6 7 8 9]
[10 11 12 13 14 15 16 17 18]  =>  [11 12 13 14 15 16 17 18 19]
[20 21 22 23 24 25 26 27 28]  =>  [21 22 23 24 25 26 27 28 29]
```

Wektoryzacja tekstu

```
from tensorflow.keras import layers
from tensorflow.keras import losses

url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

dataset = tf.keras.utils.get_file("aclImdb_v1", url,
                                  untar=True, cache_dir='.',
                                  cache_subdir='')

dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')
os.listdir(dataset_dir)
train_dir = os.path.join(dataset_dir, 'train')
os.listdir(train_dir)
```

Wektoryzacja tekstu

```
batch_size = 1024
seed = 123
train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train', batch_size=batch_size, validation_split=0.2,
    subset='training', seed=seed)
val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train', batch_size=batch_size, validation_split=0.2,
    subset='validation', seed=seed)
```

```
Found 75000 files belonging to 3 classes.
Using 60000 files for training.
Found 75000 files belonging to 3 classes.
Using 15000 files for validation.
```

```
for text_batch, label_batch in train_ds.take(1):
    for i in range(5):
        print(label_batch[i].numpy(), text_batch.numpy()[i])
```

```
1 b"Ask yourself where she got the gun? Remember what she was taught about the mark's mindset when the con
is over? The gun had blanks and it was provided to her from the very beginning.<br /><br />When the patien
t comes back at the end she was SUPPOSED to see him drive away in the red convertible and lead her to the
gang splitting up her 80 thousand.<br /><br />The patient was in on the con from the beginning.<br /><br /
>Mantegna does not die in the end - the gun had blanks.<br /><br />There - enough spoilers for you there?
This is why people are giving it such high ratings. It's extremely original because of the hidden ending a
nd how it cons MOST of the audience."
2 b"For some reason, people seem to have a problem differentiating this movie from Trail of the Pink Panth
er.<br /><br />At any rate, this work does nothing but serve to remind us how sad the world is without Pet
```

Wektoryzacja tekstu

```
batch_size = 1024
seed = 123
train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train', batch_size=batch_size, validation_split=0.2,
    subset='training', seed=seed)
val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train', batch_size=batch_size, validation_split=0.2,
    subset='validation', seed=seed)
```

```
Found 75000 files belonging to 3 classes.
Using 60000 files for training.
Found 75000 files belonging to 3 classes.
Using 15000 files for validation.
```

```
for text_batch, label_batch in train_ds.take(1):
    for i in range(5):
        print(label_batch[i].numpy(), text_batch.numpy()[i])
```

```
1 b"Ask yourself where she got the gun? Remember what she was taught about the mark's mindset when the con
is over? The gun had blanks and it was provided to her from the very beginning.<br /><br />When the patien
t comes back at the end she was SUPPOSED to see him drive away in the red convertible and lead her to the
gang splitting up her 80 thousand.<br /><br />The patient was in on the con from the beginning.<br /><br /
>Mantegna does not die in the end - the gun had blanks.<br /><br />There - enough spoilers for you there?
This is why people are giving it such high ratings. It's extremely original because of the hidden ending a
nd how it cons MOST of the audience."
2 b"For some reason, people seem to have a problem differentiating this movie from Trail of the Pink Panth
er.<br /><br />At any rate, this work does nothing but serve to remind us how sad the world is without Pet
```

Wektoryzacja tekstu

```
embedding_dim=16

model = tf.keras.Sequential([
  vectorize_layer,
  layers.Embedding(vocab_size, embedding_dim, name="embedding"),
  layers.GlobalAveragePooling1D(),
  layers.Dense(16, activation='relu'),
  layers.Dense(1)
])
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="logs")
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=15,
    callbacks=[tensorboard_callback])
model.summary()
```

```
Epoch 1/15
59/59 [==============================] - 3s 43ms/step - loss: 0.4389 - accuracy: 0.1643 - val_loss: 0.0553
```

Próbkowanie losowych tensorów

- Najczęstszym sposobem jest próbkowanie każdego wpisu w tensorze z rozkładu losowego. tf.random_normal pozwala na próbkowanie każdego wpisu w tensorze o określonym kształcie z rozkładu normalnego o określonej średniej i odchyleniu standardowym.
- **a = tf.random.normal((2, 2), mean=0, stddev=1)**

```
tf.Tensor(
[[ 0.15155038 -0.28032118]
 [ 0.6855862  -0.96309453]], shape=(2, 2), dtype=float32)
```

Zaburzenie symetrii

- Wiele algorytmów uczenia maszynowego uczy się, dokonując aktualizacji zestawu tensorów, które obsługują wagi. Te równania aktualizacji zwykle mają tę właściwość, że wagi inicjowane tą samą wartością będą ewoluować wspólnie. Jeśli więc początkowy zestaw tensorów jest inicjowany stałą wartością, model nie będzie w stanie wiele się nauczyć. Naprawienie tej sytuacji wymaga zaburzenia symetrii. Najprostszym sposobem zaburzenia symetrii jest losowe próbkowanie każdego wpisu w tensorze.

Zaburzenie symetrii

- Przy próbkowaniu dziesiątek milionów losowych wartości z rozkładu normalnego staje się niemal pewne, że niektóre próbkowane wartości będą dalekie od średniej. Tak duże próbki mogą prowadzić do niestabilności numerycznej, więc często stosuje się próbkowanie przy użyciu funkcji tf.truncated_normal() zamiast tf.random_normal(). Funkcja ta zachowuje się tak samo jak tf.random_normal(), jeśli chodzi o API, ale odrzuca i ponownie próbkuje wszystkie wartości przekraczające dwa standardowe odchylenia od średniej.

Typy tensorów

- Być może zauważyłeś w poprzednich przykładach oznaczenie dtype. W TensorFlow występują tensory różnego typu, takie jak tf.float32, tf.float64, tf.int32, tf.int64. Możliwe jest tworzenie tensorów określonego typu poprzez ustawienie argumentu dtype w funkcjach tworzenia tensora. Ponadto dla danego tensora można zmienić jego typ, używając funkcji konwertujących, takich jak tf.to_double(), tf.to_float(), tf.to_int32(), tf.to_int64() i inne.

Tensor broadcasting

- Rozgłaszanie (ang. broadcasting) to termin (wprowadzony przez NumPy) stosowany w przypadku, gdy można dodać do siebie macierze i wektory systemu tensorów o różnych rozmiarach. Reguły te pozwalają na takie udogodnienia jak dodawanie wektora do każdego rzędu macierzy. Zasady rozgłaszania mogą być dość złożone, więc nie będziemy wdawać się w formalną dyskusję na ten temat. Często łatwiej jest eksperymentować i samemu zobaczyć, jak działa rozgłaszanie.

Tensor broadcasting - przykład nr 1 (ten sam shape)

Tensor 1:

```
[[1, 2, 3],]

rank: 2
shape: (1,3)
```

Tensor 2:

```
[[4, 5, 6],]

rank: 2
shape: (1,3)
```

Tensor 1 + Tensor 2:

```
    [[1, 2, 3],]
+
    [[4, 5, 6],]
--------------------
    [[5, 7, 9],]
    rank: 2
    shape: (1,3)
```

Tensor broadcasting - przykład nr 2 (różny shape, ten sam rank)

Tensor 1:

```
[[1, 2, 3],]

rank: 2
shape: (1,3)
```

Tensor 2:

```
[[4],
 [5],
 [6]]

rank: 2
shape: (3,1)
```

Tensor 1 + Tensor 2:

```
    [[1, 2, 3],]
+
    [[4],
     [5],
     [6]]
--------------------
    [[5, 6, 7],
     [6, 7, 8],
     [7, 8, 9]]
    rank: 2
    shape: (3,3)
```

Broadcasting tensora 1 do shape(3,3)

Tensor 1 Broadcast To Shape (3,3):

```
Before:
    [[1, 2, 3],]

After:
    [[1, 2, 3],
     [1, 2, 3],
     [1, 2, 3]]
```

Broadcasting tensora 2 do shape(3,3)

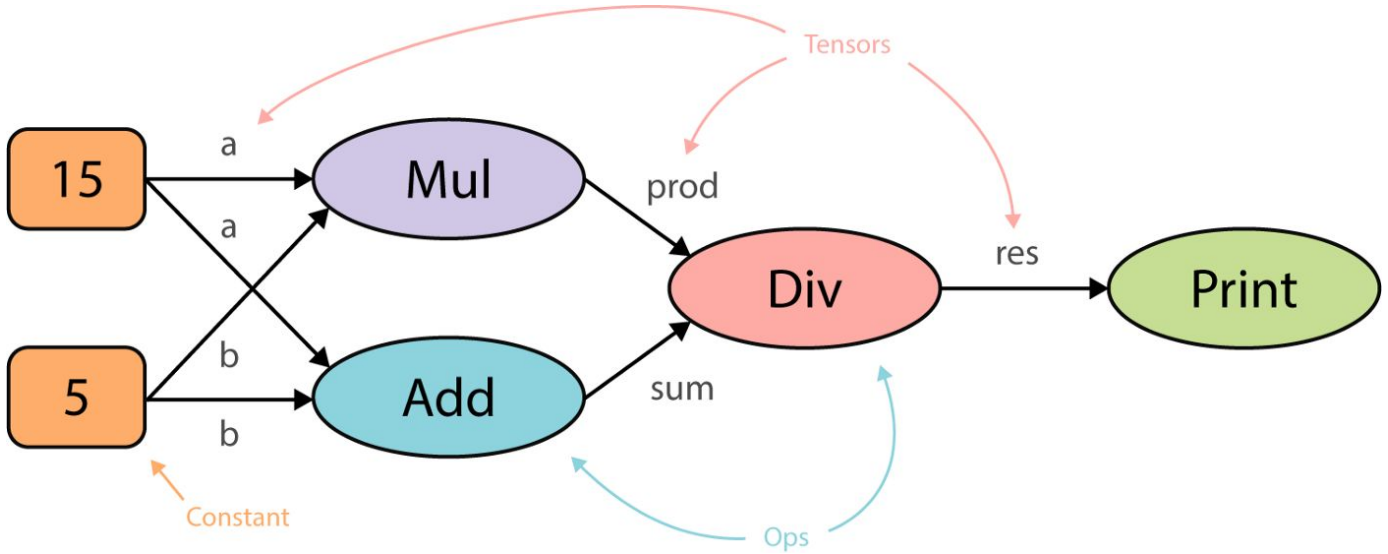
```
Before:
    [[4],
     [5],
     [6]]

After:
    [[4, 4, 4],
     [5, 5, 5],
     [6, 6, 6]]
```

Wykres obliczeniowy

```
a = 15
b = 5
prod = a * b
sum = a + b
res = prod / sum
print(res)
```

```
3.75
```

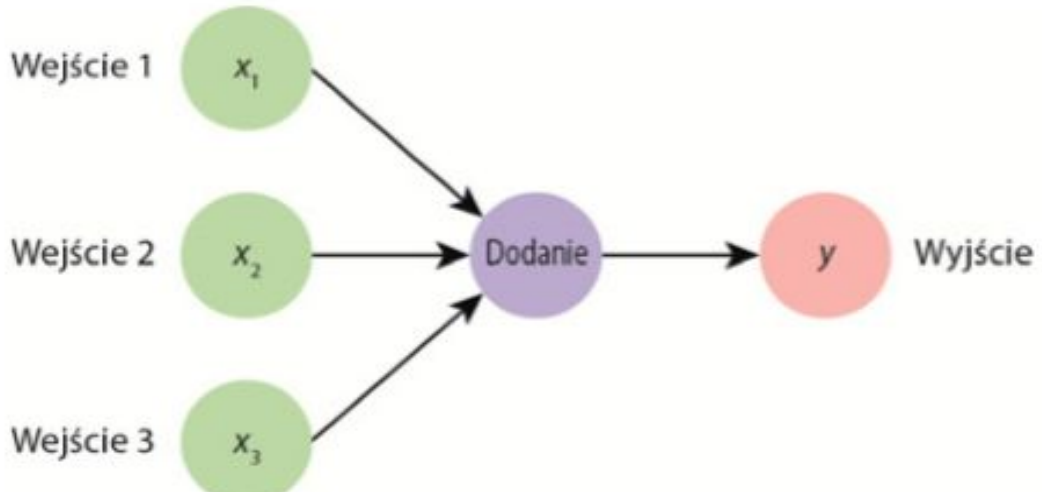


Graf obliczeniowy

Wykres obliczeń jest podstawową jednostką obliczeń w TensorFlow.
Model w TensorFlow zawiera graf wyliczeniowy.

Innymi słowy grafy są strukturami danych, które zawierają zbiór obiektów tf.Operation, które reprezentują jednostki obliczeniowe oraz obiekty tf.Tensor, które reprezentują jednostki danych przepływające pomiędzy operacjami.
tf.Graph - kontekst grafu przepływu danych

Definiowanie grafu obliczeniowego z 3 wejściami (y = x1+x2+x3)



Wykres obliczeniowy

```
import tensorflow as tf
import timeit
from datetime import datetime
```

```
def a_regular_function(x, y, b):
    x = tf.matmul(x, y)
    x = x + b
    return x

a_function_that_uses_a_graph = tf.function(a_regular_function)

x1 = tf.constant([[1.0, 2.0]])
y1 = tf.constant([[2.0], [3.0]])
b1 = tf.constant(4.0)

orig_value = a_regular_function(x1, y1, b1).numpy()
tf_function_value = a_function_that_uses_a_graph(x1, y1, b1).numpy()
assert(orig_value == tf_function_value)
print(orig_value)
```

```
[[12.]]
```

@tf.function

```
@tf.function
def simple_relu(x):
    if tf.greater(x, 0):
        return x
    else:
        return 0

print("First branch, with graph:", simple_relu(tf.constant(1)).numpy())
print("Second branch, with graph:", simple_relu(tf.constant(-1)).numpy())
```

```
First branch, with graph: 1
Second branch, with graph: 0
```

```
type(simple_relu)
```

```
tensorflow.python.eager.def_function.Function
```

@tf.autograph

```
print(tf.autograph.to_code(simple_relu.python_function))
```

```
def tf__simple_relu(x):
    with ag__.FunctionScope('simple_relu', 'fscope', ag__.ConversionOptions(recursive=True, user_r
equested=True, optional_features=(), internal_convert_user_code=True)) as fscope:
        do_return = False
        retval_ = ag__.UndefinedReturnValue()

        def get_state():
            return (do_return, retval_)

        def set_state(vars_):
            nonlocal retval_, do_return
            (do_return, retval_) = vars_

        def if_body():
            nonlocal retval_, do_return
            try:
                do_return = True
                retval_ = ag__.ld(x)
            except:
                do_return = False
                raise

        def else_body():
            nonlocal retval_, do_return
            try:
                do_return = True
                retval_ = 0
            except:
                do_return = False
                raise
        ag__.if_stmt(ag__.converted_call(ag__.ld(tf).greater, (ag__.ld(x), 0), None, fscope), if_b
ody, else_body, get_state, set_state, ('do_return', 'retval_'), 2)
        return fscope.ret(retval_, do_return)
```

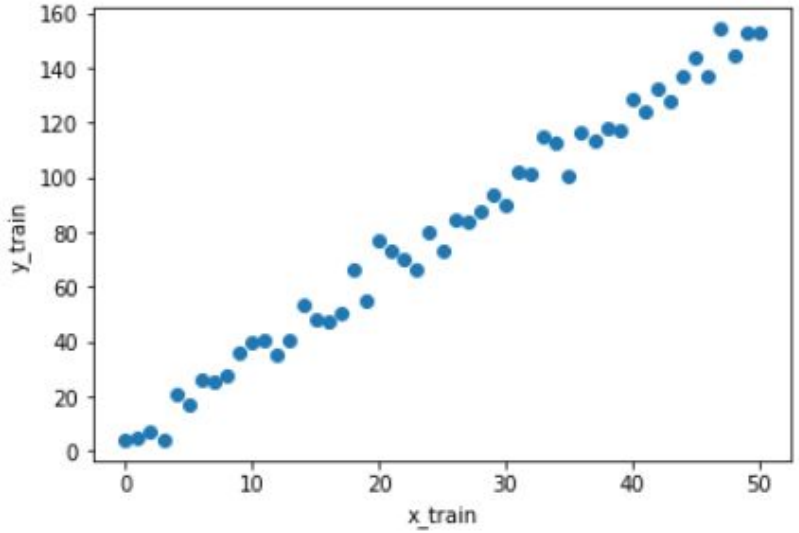
Regresja liniowa Sekwencyjna API

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

x_train = np.linspace(0, 50, 51)
y_train = np.linspace(5, 155, 51)
```

```
# add noise
y_train = y_train + np.random.normal(0,5,51)
```

```
plt.xlabel('x_train')
plt.ylabel('y_train')
plt.scatter(x_train, y_train)
plt.show()
```

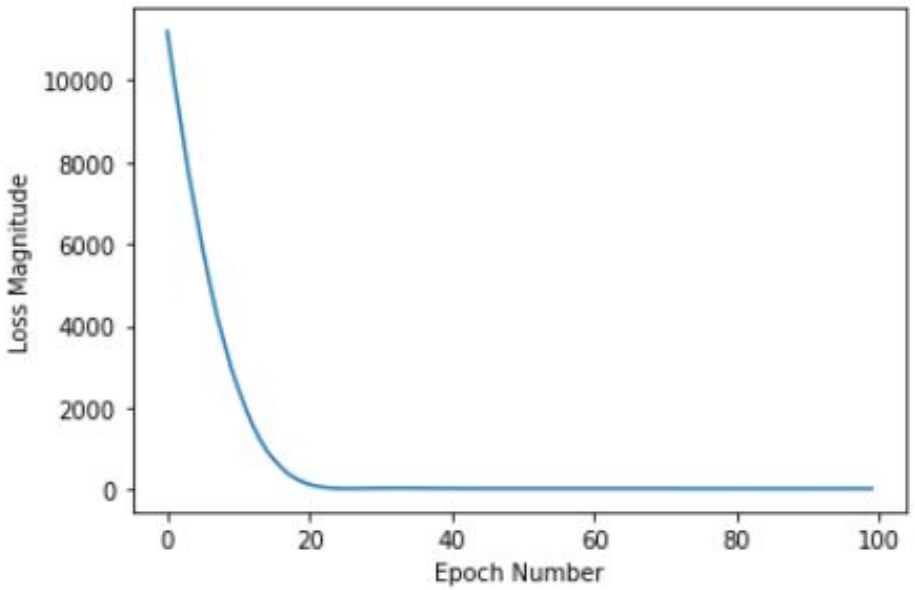


Regresja liniowa Sekwencyjna API

```
# build linear model
layer0 = tf.keras.layers.Dense(units=1, input_shape=[1])
model = tf.keras.Sequential([layer0])
model.compile(loss='mean_squared_error',
              optimizer=tf.keras.optimizers.Adam(0.1))

history = model.fit(x_train,y_train, epochs=100, verbose=False)
```

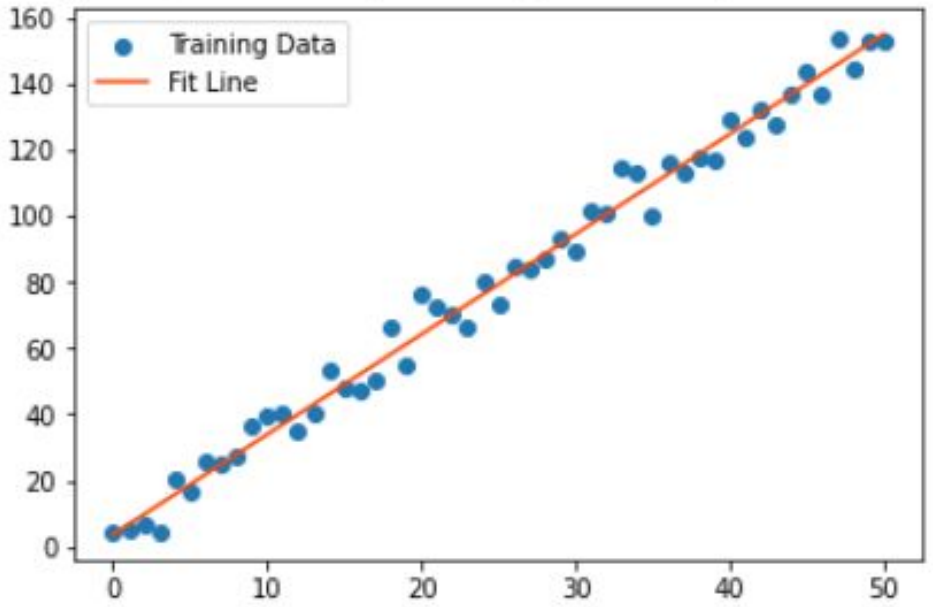
```
plt.xlabel('Epoch Number')
plt.ylabel("Loss Magnitude")
plt.plot(history.history['loss'])
plt.show()
```



Regresja liniowa Sekwencyjna API

```
weights = layer0.get_weights()
weight = weights[0][0]
bias = weights[1]
print('weight: {} bias: {}'.format(weight, bias))
y_learned = x_train * weight + bias
plt.scatter(x_train, y_train, label='Training Data')
plt.plot(x_train, y_learned, color='orangered', label='Fit Line')
plt.legend()
plt.show()
```

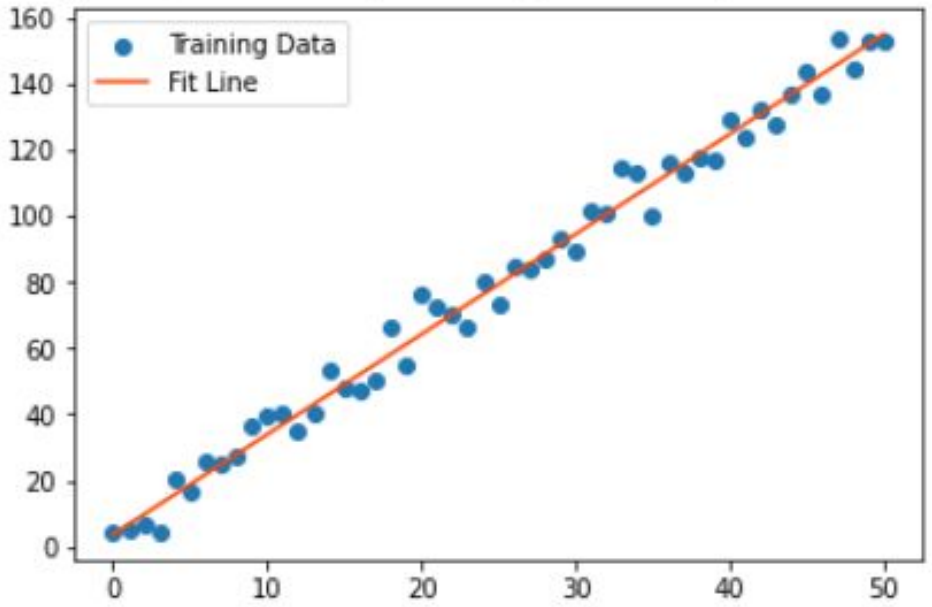
```
weight: [3.0297072] bias: [3.6556325]
```



Regresja liniowa Sekwencyjna API

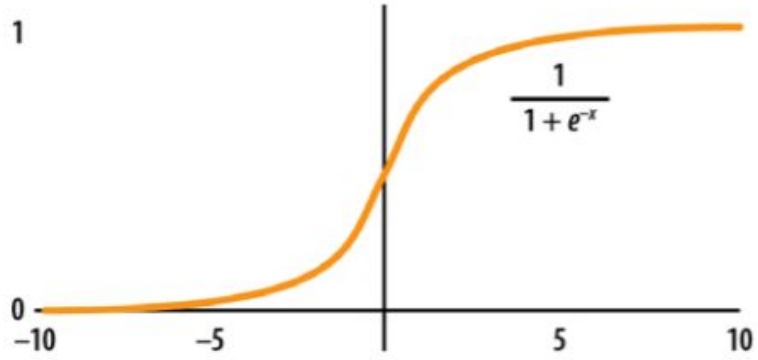
```
weights = layer0.get_weights()
weight = weights[0][0]
bias = weights[1]
print('weight: {} bias: {}'.format(weight, bias))
y_learned = x_train * weight + bias
plt.scatter(x_train, y_train, label='Training Data')
plt.plot(x_train, y_learned, color='orangered', label='Fit Line')
plt.legend()
plt.show()
```

```
weight: [3.0297072] bias: [3.6556325]
```



Regresja logistyczna w tensorflow

- Warto najpierw zastanowić się, czym jest równanie dla klasyfikatora. Powszechnie stosowanym trikiem matematycznym jest wykorzystanie funkcji sigmoidalnej. Sigmoida wykreślona na poniższym rysunku, powszechnie oznaczana symbolem σ, jest funkcją przekształcającą liczby rzeczywiste ℝ do zakresu (0, 1). Ta właściwość jest przydatna, ponieważ możemy interpretować wyjście sigmoidy jako prawdopodobieństwo zajścia zdarzenia (przekształcanie zdarzeń dyskretnych w wartości ciągłe to powracający motyw w uczeniu maszynowym).



Regresja logistyczna z MNIST

```
# MNIST dataset parameters.
num_classes = 10 # 0 to 9 digits
num_features = 784 # 28*28

# Training parameters.
learning_rate = 0.01
training_steps = 1000
batch_size = 256
display_step = 50
```

```
# Prepare MNIST data.
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Convert to float32.
x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
# Flatten images to 1-D vector of 784 features (28*28).
x_train, x_test = x_train.reshape([-1, num_features]), x_test.reshape([-1, num_features])
# Normalize images value from [0, 255] to [0, 1].
x_train, x_test = x_train / 255., x_test / 255.
```

```
# Use tf.data API to shuffle and batch data.
train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_data = train_data.repeat().shuffle(5000).batch(batch_size).prefetch(1)
```

Regresja logistyczna z MNIST

```
# Use tf.data API to shuffle and batch data.
train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_data = train_data.repeat().shuffle(5000).batch(batch_size).prefetch(1)
```

```
# Weight of shape [784, 10], the 28*28 image features, and total number of classes.
W = tf.Variable(tf.ones([num_features, num_classes]), name="weight")
# Bias of shape [10], the total number of classes.
b = tf.Variable(tf.zeros([num_classes]), name="bias")
```

```
# Weight of shape [784, 10], the 28*28 image features, and total number of classes.
W = tf.Variable(tf.ones([num_features, num_classes]), name="weight")
# Bias of shape [10], the total number of classes.
b = tf.Variable(tf.zeros([num_classes]), name="bias")

# Logistic regression (Wx + b).
def logistic_regression(x):
    # Apply softmax to normalize the logits to a probability distribution.
    return tf.nn.softmax(tf.matmul(x, W) + b)
```

Regresja logistyczna z MNIST

```python
# Cross-Entropy loss function.
def cross_entropy(y_pred, y_true):
    # Encode label to a one hot vector.
    y_true = tf.one_hot(y_true, depth=num_classes)
    # Clip prediction values to avoid log(0) error.
    y_pred = tf.clip_by_value(y_pred, 1e-9, 1.)
    # Compute cross-entropy.
    return tf.reduce_mean(-tf.reduce_sum(y_true * tf.math.log(y_pred),1))
```

```python
# Accuracy metric.
def accuracy(y_pred, y_true):
    # Predicted class is the index of highest score in prediction vector (i.e. argmax).
    correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true, tf.int64))
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```python
optimizer = tf.optimizers.SGD(learning_rate)
```

```python
def run_optimization(x, y):
    with tf.GradientTape() as g:
        pred = logistic_regression(x)
        loss = cross_entropy(pred, y)

    gradients = g.gradient(loss, [W, b])
    optimizer.apply_gradients(zip(gradients, [W, b]))
```

Regresja logistyczna z MNIST

```
# training loop
for step, (batch_x, batch_y) in enumerate(train_data.take(training_steps), 1):
    run_optimization(batch_x, batch_y)

    if step % display_step == 0:
        pred = logistic_regression(batch_x)
        loss = cross_entropy(pred, batch_y)
        acc = accuracy(pred, batch_y)
        print("step: %i, loss: %f, accuracy: %f" % (step, loss, acc))
```

```
step: 50, loss: 1.859208, accuracy: 0.714844
step: 100, loss: 1.579595, accuracy: 0.730469
step: 150, loss: 1.371134, accuracy: 0.812500
step: 200, loss: 1.132960, accuracy: 0.835938
```

Regresja logistyczna z MNIST

```
pred = logistic_regression(x_test)
print("Test Accuracy: %f" % accuracy(pred, y_test))
```

```
Test Accuracy: 0.870300
```

```
# Predict 5 images from validation set.
n_images = 5
test_images = x_test[:n_images]
predictions = logistic_regression(test_images)

# Display image and model prediction.
for i in range(n_images):
    plt.imshow(np.reshape(test_images[i], [28, 28]), cmap='gray')
    plt.show()
    print("Model prediction: %i" % np.argmax(predictions.numpy()[i]))
```

MNIST z siecią neuronową

```
# load tensorboard extension
%load_ext tensorboard
```

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train),(x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
def create_model():
    return tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
  ])
```

MNIST z siecią neuronową

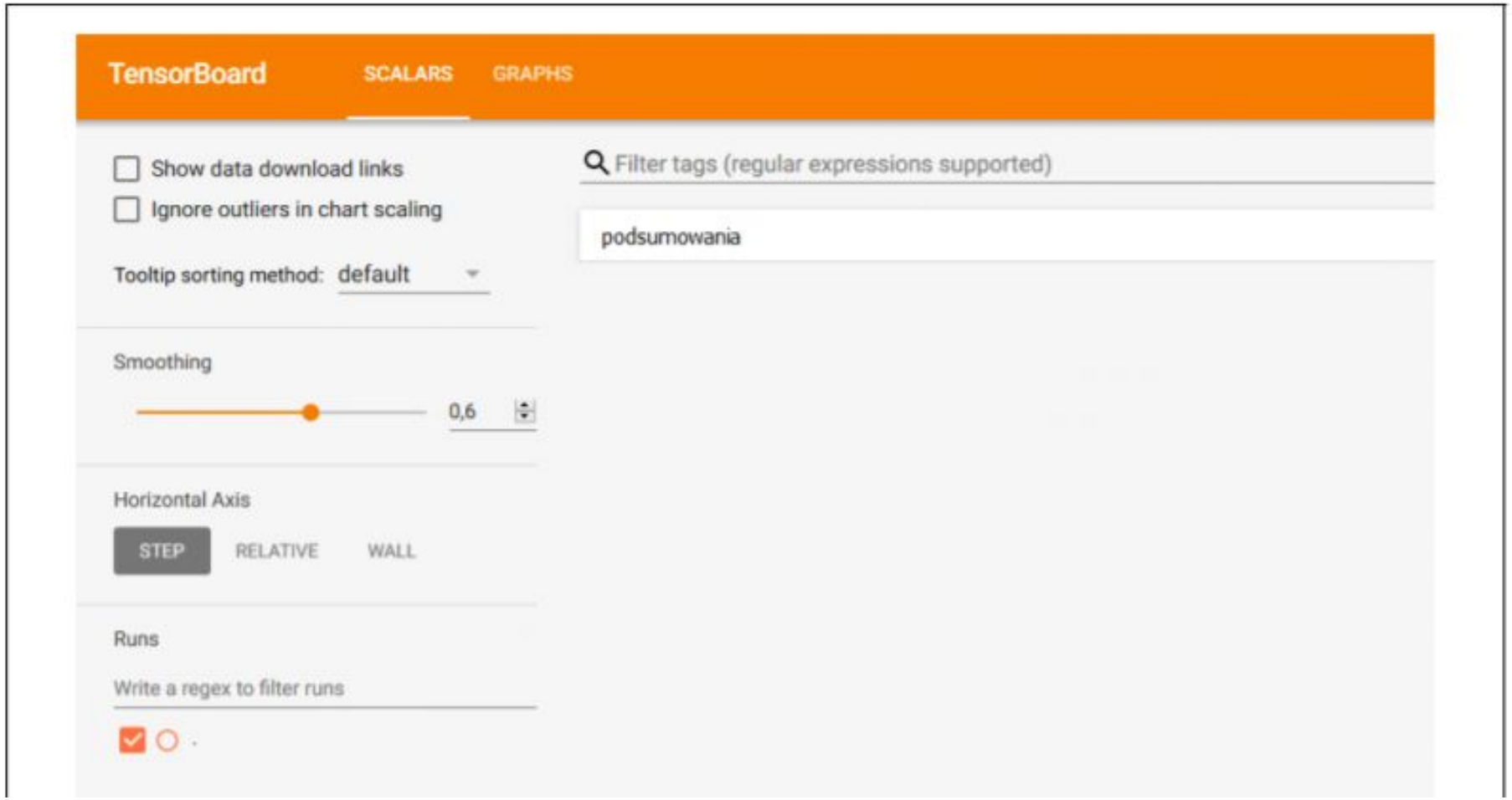
```
def train_model():
    model = create_model()
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
    tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

    model.fit(x=x_train,
              y=y_train,
              epochs=5,
              validation_data=(x_test, y_test),
              callbacks=[tensorboard_callback])

train_model()
```

Zrzut ekranu tensorboard

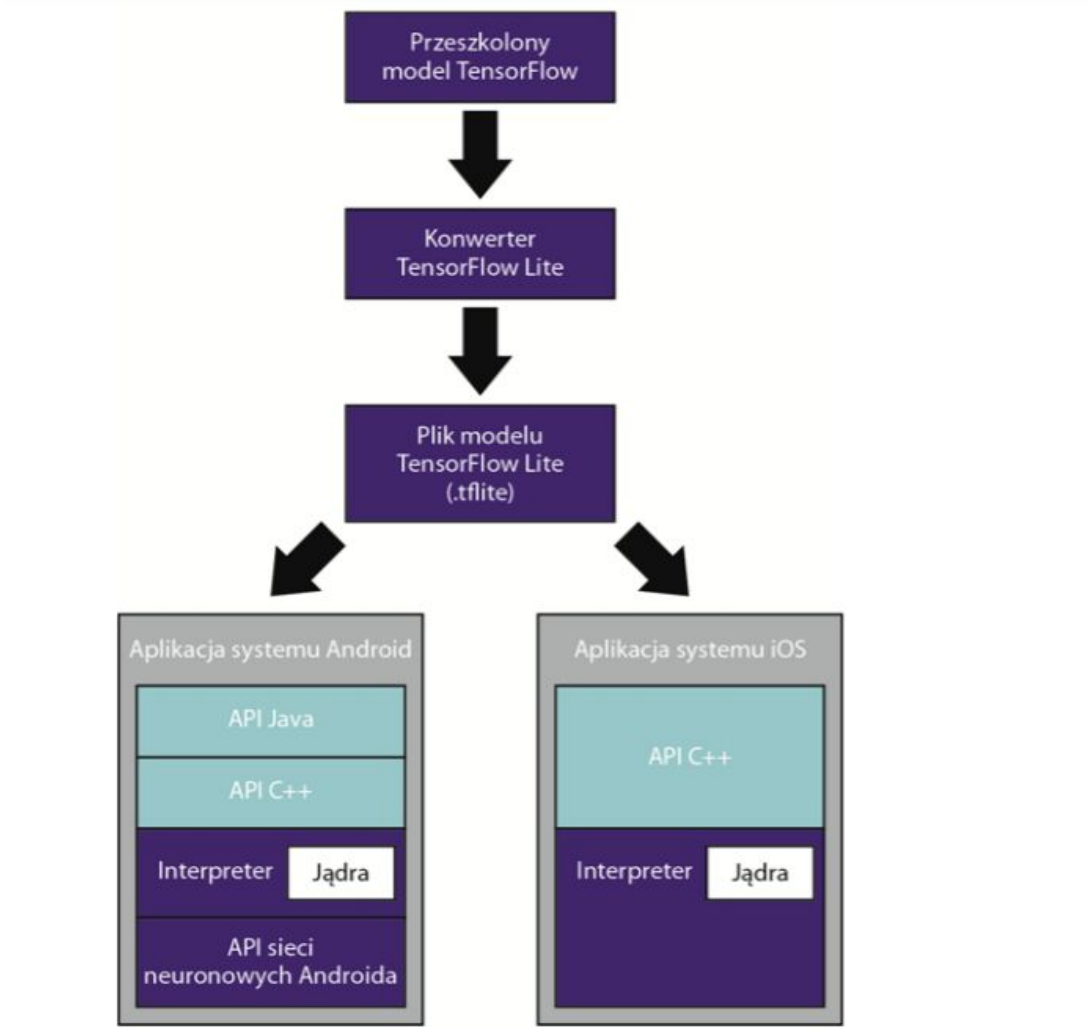


Tensorflow Lite

Zalety używania aplikacji tensorflow na urządzeniach mobilnych:

Prywatność — jeśli trenowanie modelu ML może być przeprowadzone na urządzeniu
Prognozowanie lub klasyfikacja w trybie offline
Rozwój inteligentnych urządzeń
Systemy oparte o Machine Learning mogą być bardziej energooszczędne
Wykorzystanie danych z urządzeń Internet of Things

Architektura tensorflow lite



Polecane materiały związane z tematyką bloku

- https://www.tensorflow.org/tutorials
- https://www.tutorialspoint.com/tensorflow/index.htm
- https://adventuresinmachinelearning.com/python-tensorflow-tutorial