Biblioteka PyTorch

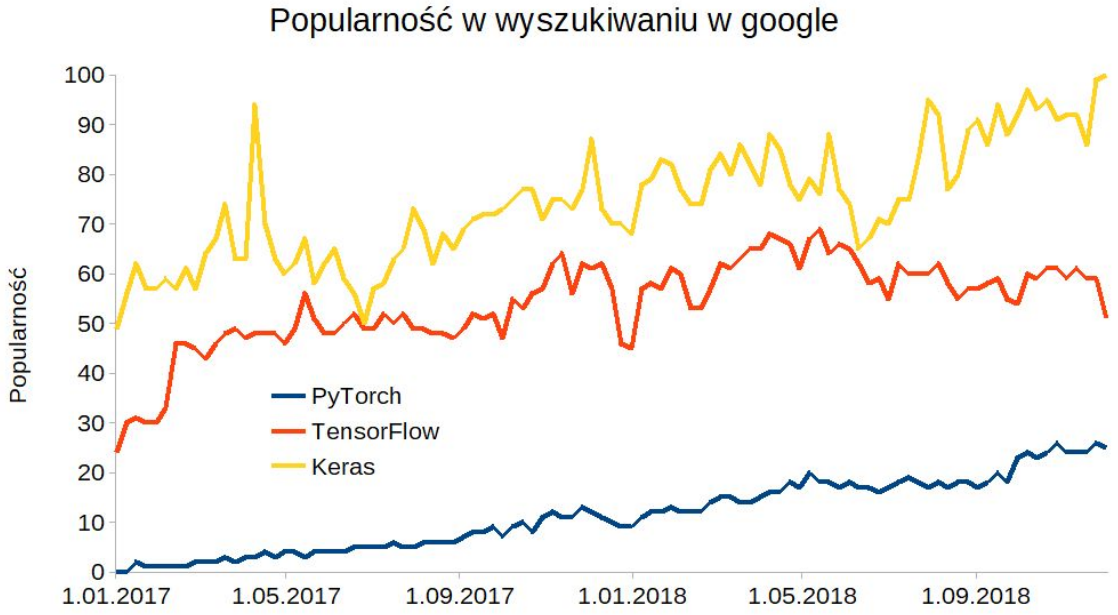
Filip Andrzejewski
Piotr Jaszkiewicz

Co daje PyTorch

- Obliczenia na tensorach przy wsparciu karty graficznej
- Uczenie głębokich sieci neuronowych
- Obecnie w wersji beta

Cechy

- Sieć neuronowa jest dynamiczna
- Po prostu Python
- Imperatywny
- Łatwy w rozszerzaniu



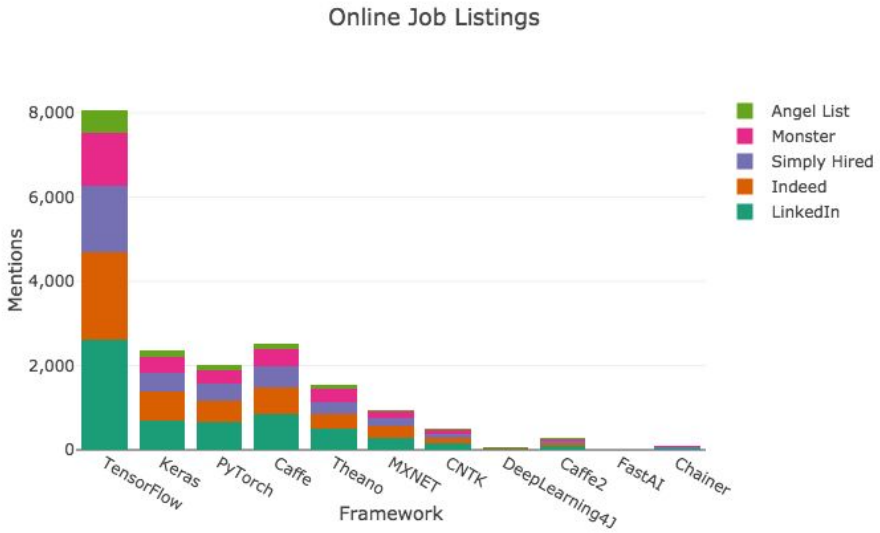
wykres na podstawie trends.google.pl/

Kto to rozwija

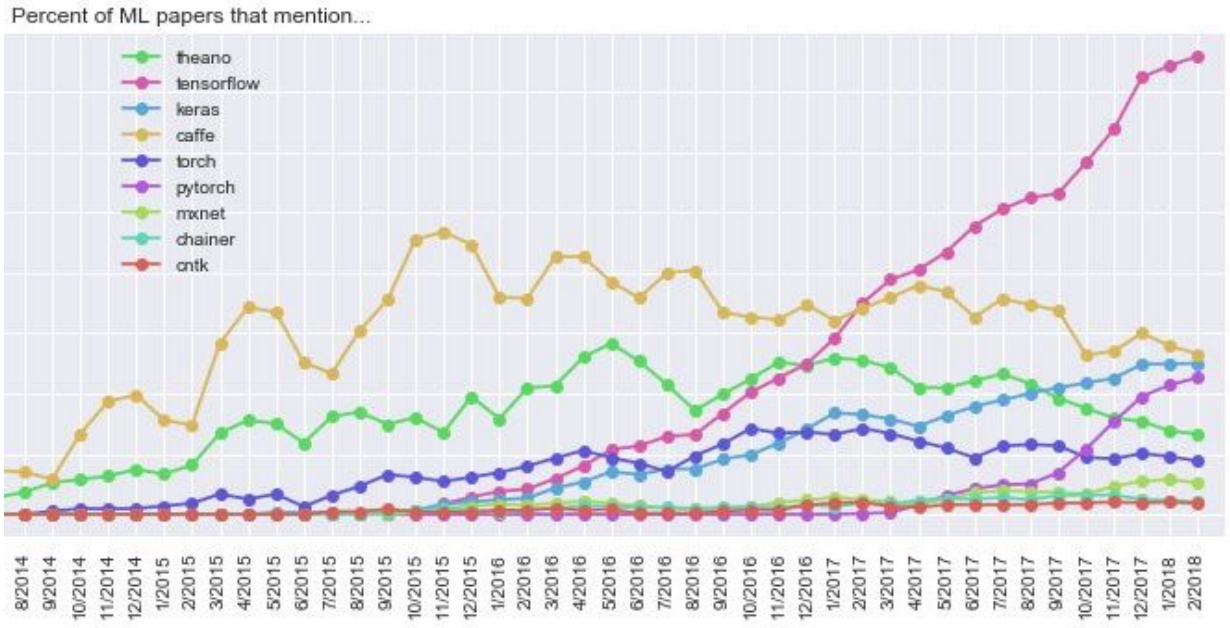
- Stworzony na podstawie biblioteki Torch
- Utrzymywany przez Facebooka
- Open-source (BSD-like)

Porównanie z Tensorflow/Keras

- Programowanie symboliczne vs. programowanie imperatywne
- Statyczny graf vs. dynamiczny (rozwijanie rNN i używanie tf.while_loop())
- Tensorflow i Keras są bardziej popularne – więcej materiałów



Zastosowania



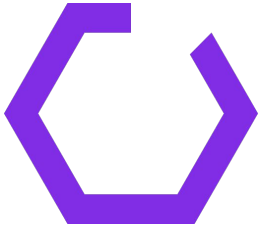
https://twitter.com/karpathy/status/972295865187512320

Zastosowania

https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

Projekty wykorzystujące PyTorch

http://demo.allennlp.org



Glow

Tensory

Tensory w PyTorch to wielowymiarowe macierze zawierające elementy jednego typu (najczęściej float).

Numpy

```
x1 = np.linspace( 2, 100, 50)
x2 = np.sin(x1)
x3 = x1 + x2
x4 = np.matmul(x1.reshape( 1,-1), x1.reshape(- 1,
1))

x5 = torch.from_numpy(x1)
```

PyTorch

```
y1 = torch.linspace( 2, 100, 50)
y2 = torch.sin(y1)
y3 = y1 + y2
y4 = torch.mm(y1.view( 1,-1), y1.view(- 1,1))

y5 = y1.numpy()
```

Automatyczne różniczkowanie

Tensor można opakować w Variable, który zapamiętuje operacje wykonane na tym tensorze, gradient z nim związany oraz funkcje które były dla niego wejściem.

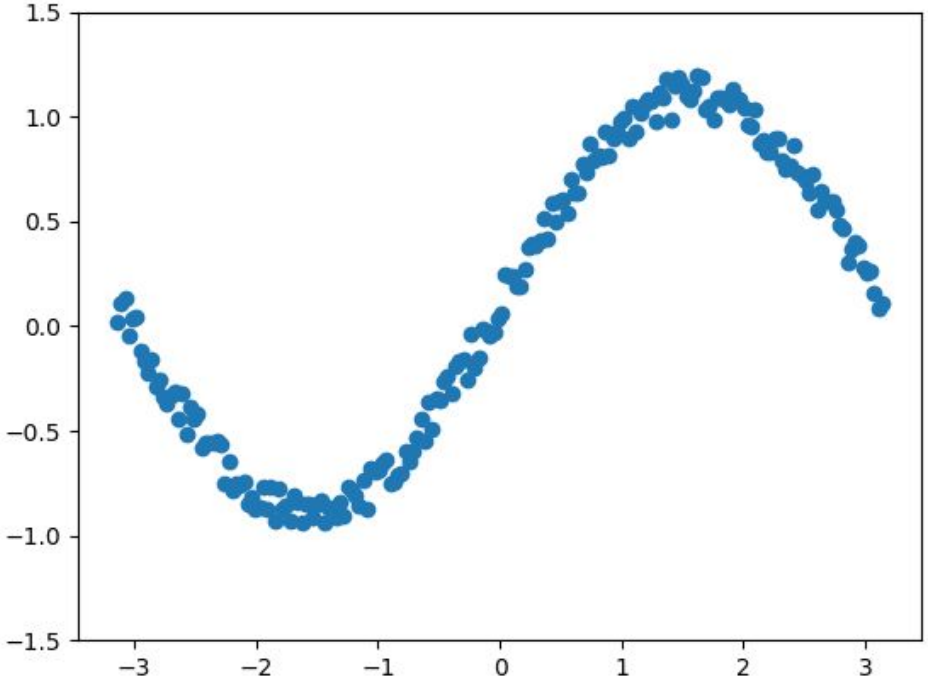
```
x = Variable(torch.ones( 2, 2), requires_grad=True)
```

Imperatywność

- Ułatwia debugowanie
- Pozwala użyć dowolnej biblioteki do tworzenia wykresów

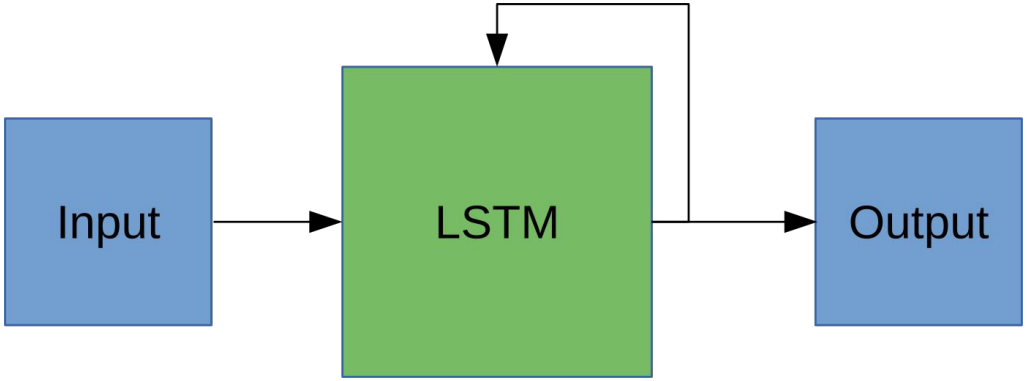
Demo

Prosty przykład regresji



Demo

Long short-term memory



Demo

Jak przenieść się na GPU

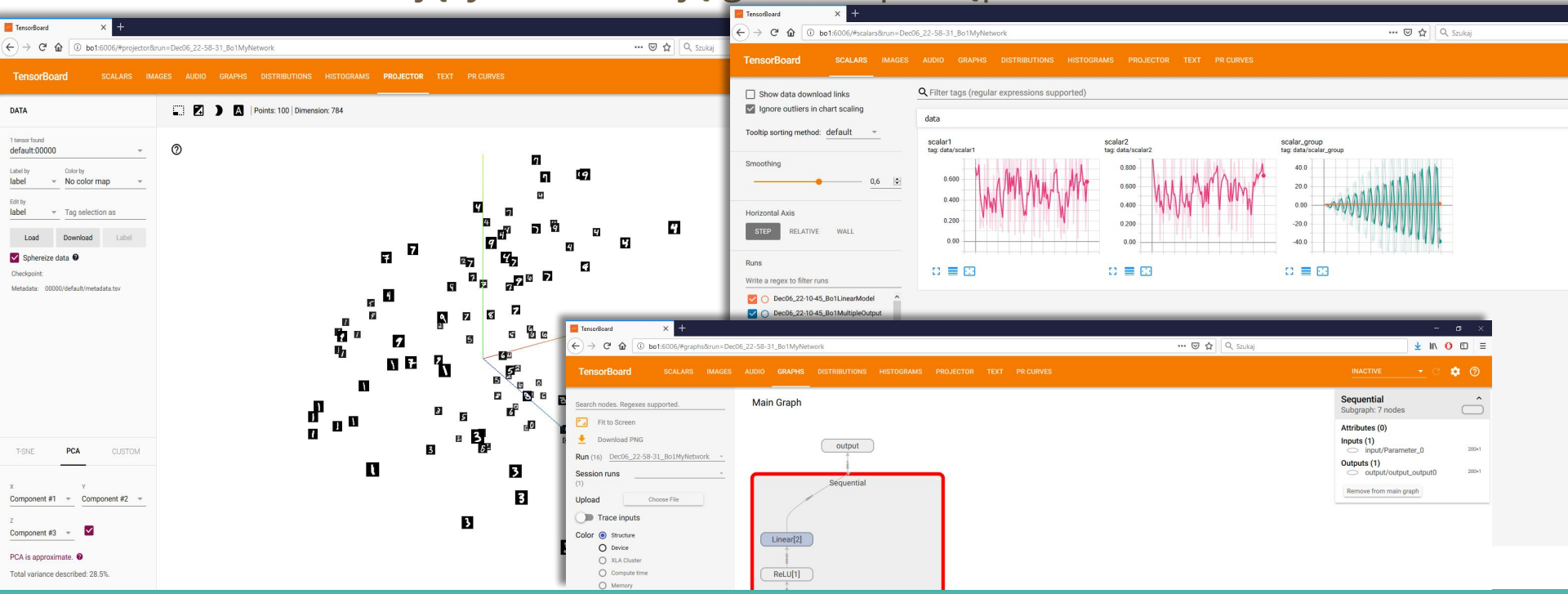
- Brak jednej flagi umożliwiającej wybór procesora jak w TensorFlow:
  `tf.device('/gpu:0')` i `tf.device('/cpu:0')`
- Można za to liczyć jednocześnie na wielu CPU i GPU.

1. Przesłanie modelu (parametrów, wag itp.) na GPU:
   `my_module.cuda()`
2. Przesyłanie tensorów na GPU:
   `x_gpu = x.cuda()`

Demo

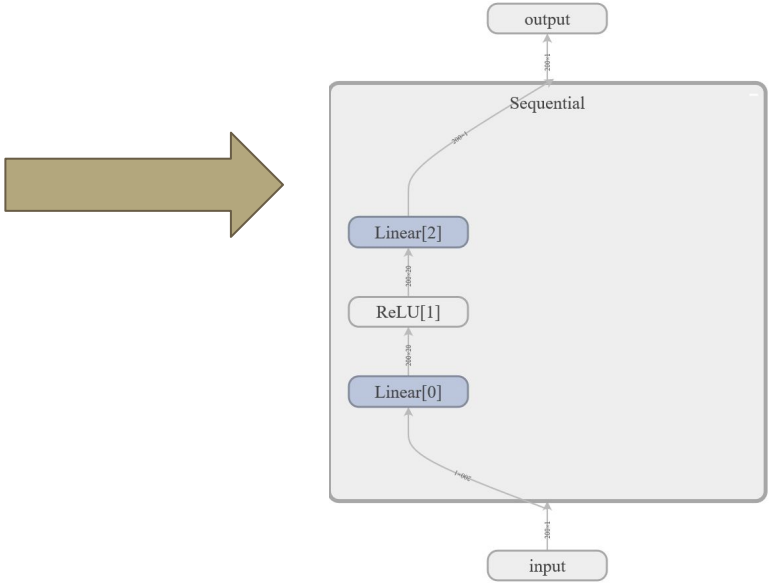
tensorboardX

Dodatek umożliwiający wizualizację grafów i postępów nauki.



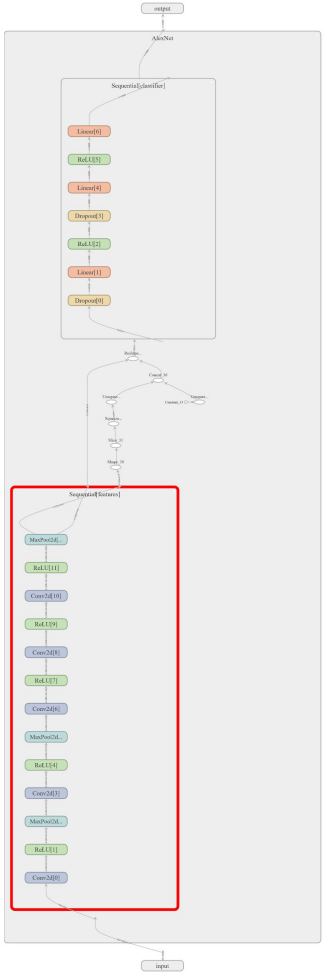
tensorboardX

```
from tensorboardX import SummaryWriter
[...]
with SummaryWriter(comment='MyNetwork') as w:
    w.add_graph(net, x)zz
```



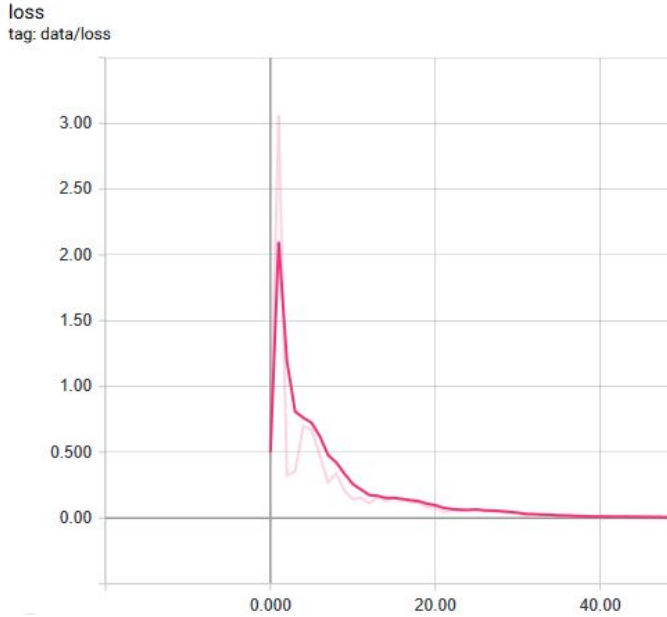
tensorboardX

```
import torch
import torchvision
from tensorboardX  import SummaryWriter
from torch.autograd  import Variable

dummy_input = Variable(torch.rand( 13, 1, 28, 28))
with SummaryWriter(comment='alexnet') as w:
    model = torchvision.models.alexnet()
    w.add_graph(model, (dummy_input,))
```



tensorboardX

```
from tensorboardX import SummaryWriter
[...]
writer = SummaryWriter(comment='Regression')
for t in range(200):
    prediction = net(x)
    loss = loss_func(prediction, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    writer.add_scalar('data/loss', loss.data.numpy(), t)
```



JIT

Po przygotowaniu sieci można ją zamienić na Torch Script, który może być skompilowany i użyty np. w C++.

Demo

JIT

```
#include <torch/script.h> // One-stop header.

#include <iostream>
#include <memory>

int main(int argc, const char* argv[]) {
  // Deserialize the ScriptModule from a file using torch::jit::load().
  std::shared_ptr<torch::jit::script::Module> module = torch::jit::load("test");
  // Create a vector of inputs.
  std::vector<torch::jit::IValue> inputs;
  inputs.push_back(torch::ones({1, 3, 224, 224}));

  // Execute the model and turn its output into a tensor.
  at::Tensor output = module->forward(inputs).toTensor();
  std::cout << output.slice(/*dim=*/1, /*start=*/0, /*end=*/5) << '\n';
}
```

Źródła

https://github.com/pytorch/pytorch

https://www.youtube.com/watch?v=nbJ-2G2GXL0

https://github.com/MorvanZhou/PyTorch-Tutorial

https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Dziękujemy