Deep Learning & Pytorch

2017.07.15

최건호

GunhoChoi / Kind_PyTorch_Tutorial

Unwatch 4 | Star 108 | Fork 18

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Settings | Insights

Kind PyTorch Tutorial for beginners

Edit

pytorch-tutorial pytorch python deep-learning Manage topics

184 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

GunhoChoi committed on GitHub Update Variable_Autograd.ipynb — Latest commit f6ce444 8 days ago

01_Math_Basics	Rename 1_Math_Basics/Math_Basics.ipynb to 01_Math_Basics/Math_Basics....	2 months ago
02_Matrix_Basics	Rename 2_Matrix_Basics/Matrix_Basics.ipynb to 02_Matrix_Basics/Matrix...	2 months ago
03_Variable_Autograd	Update Variable_Autograd.ipynb	8 days ago
04_Linear_Regression	Rename 4_Linear_Regression/Linear_Regression.ipynb to 04_Linear_Regre...	2 months ago
05_MNIST_CNN	Update MNIST_CNN.ipynb	9 days ago
06_Input_Pipeline_Pretrained	Rename 6_Input_Pipeline_Pretrained/Input_Pipeline_Pretrained.ipynb to...	2 months ago
07_Autoencoder_Model_Save	Create readme.md	2 months ago
08_Denoising_Autoencoder	Add files via upload	2 months ago
09_Simple_Char_RNN	Rename 9_Simple_Char_RNN/Simple_Char_RNN.py to 09_Simple_Char_RNN/Sim...	2 months ago
10_GAN_LayerName_MultiGPU	Update GAN_LayerName_MultiGPU.py	2 months ago
11_InfoGAN_Least_Squares_Loss	Add files via upload	2 months ago
12_StyleTransfer_ResNet	Update 12_StyleTransfer_LBFGS_gpu.ipynb	2 months ago
13_Semantic_Segmentation	Update main.py	9 days ago
logo	Delete p	2 months ago
README.md	Update README.md	15 days ago

INDEX

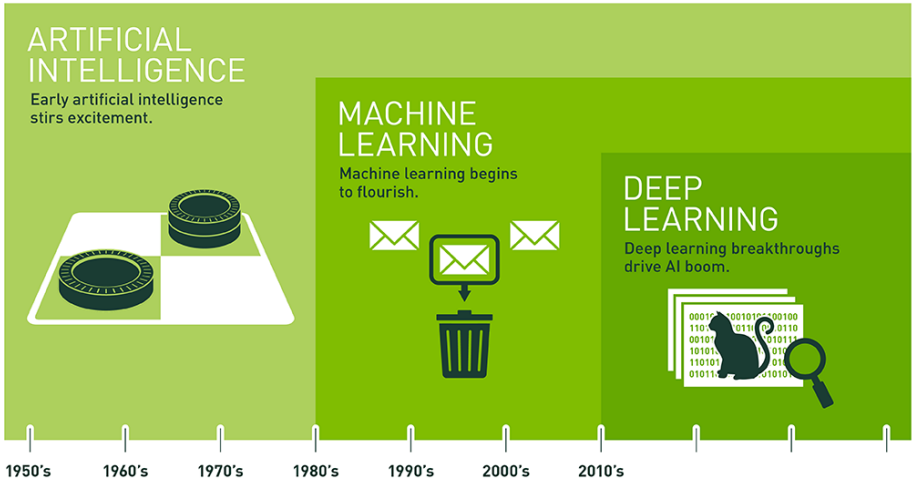
01 Deep Learning
- 정의
- 이유
- 활용

02 PyTorch
- 프레임워크 소개
- 비교

03 Installation
- CUDA
- Cudnn
- PyTorch

04 Packages
- torch.Tensor
- torch.nn
- torch.optim
- torch.autograd

Deep Learning



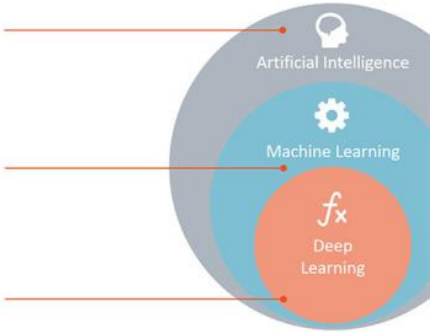
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

(출처: 엔비디아 블로그)

Artificial Intelligence
Any technique which enables computers to mimic human behavior.

Machine Learning
Subset of AI techniques which use statistical methods to enable machines to improve with experiences.

Deep Learning
Subset of ML which make the computation of multi-layer neural networks feasible.

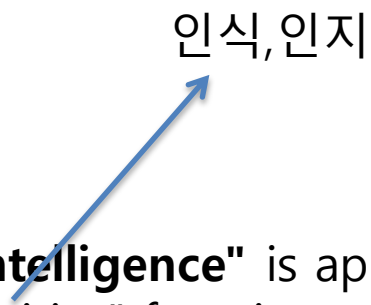


(출처: rapid miner)

Artificial Intelligence

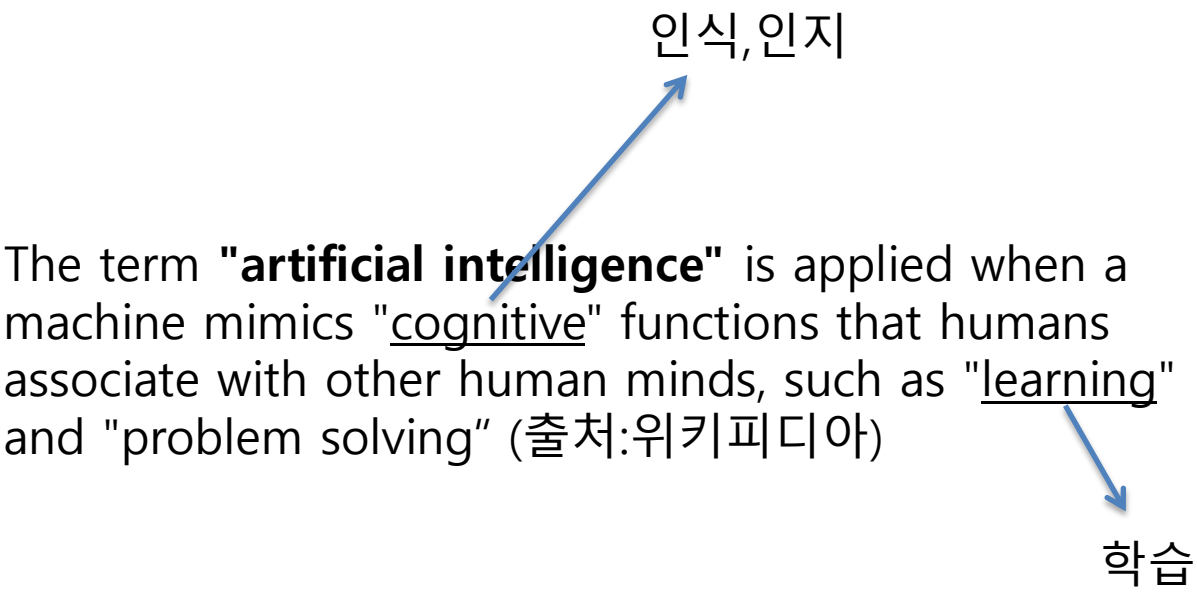
The term **"artificial intelligence"** is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving" (출처:위키피디아)

Artificial Intelligence

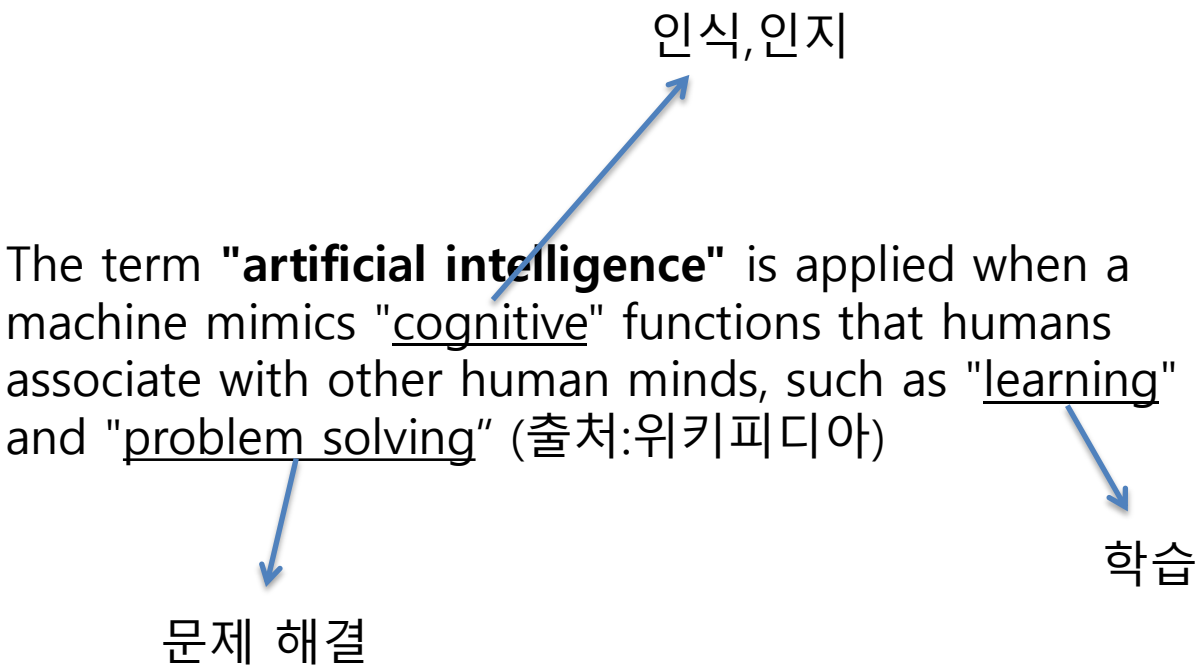


The term **"artificial intelligence"** is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving" (출처:위키피디아)

Artificial Intelligence



Artificial Intelligence



Artificial Intelligence

인식,인지 (정보 -> 지식)

The term **"artificial intelligence"** is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving" (출처:위키피디아)

학습

문제 해결

Machine Learning

Machine learning is the subfield of computer science that, according to Arthur Samuel in 1959, gives "computers the ability to learn without being explicitly programmed." (출처:위키피디아)

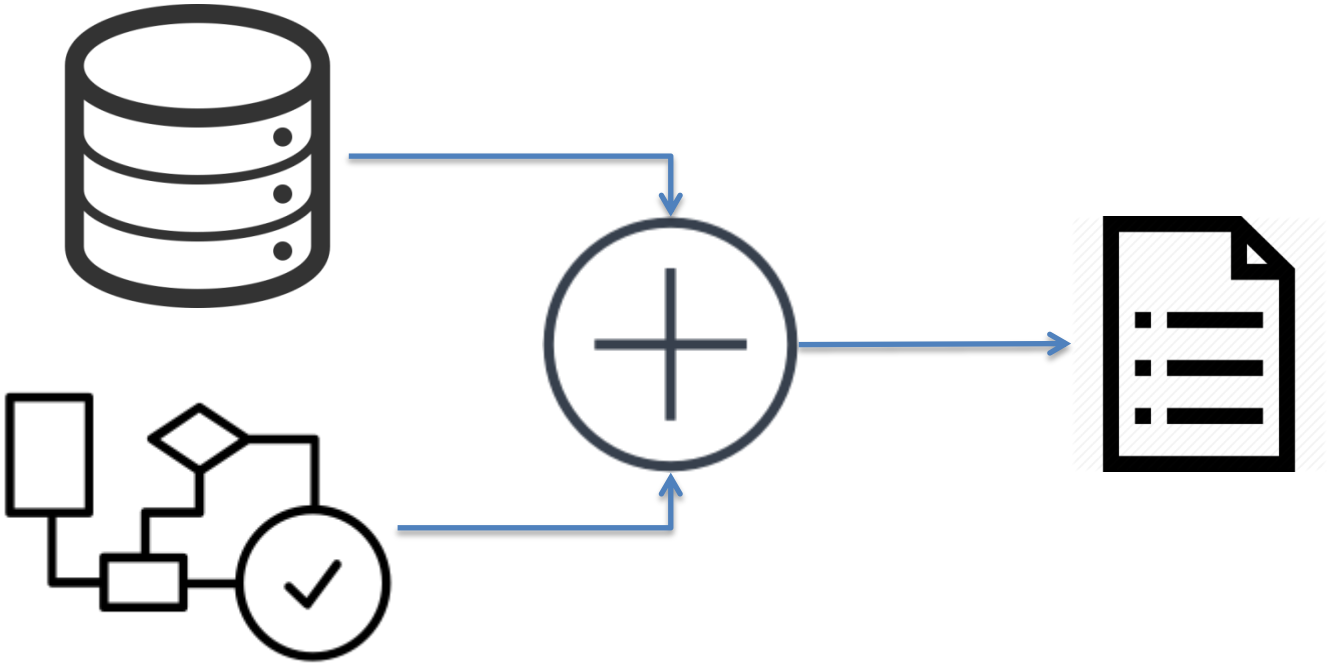
Machine Learning

Machine learning is the subfield of computer science that, according to Arthur Samuel in 1959, gives "computers the ability to learn without being explicitly programmed." (출처:위키피디아)

명시적 프로그래밍 X

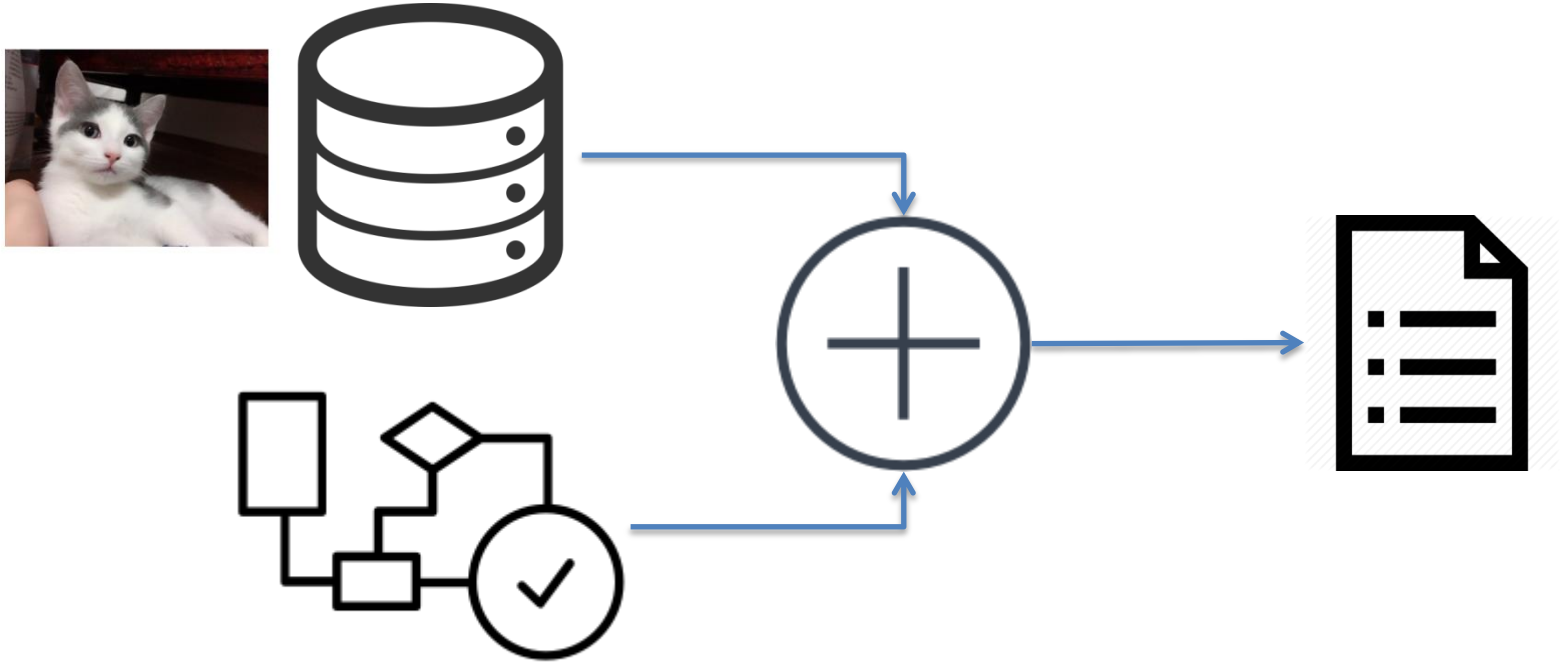
Machine Learning

명시적 프로그래밍



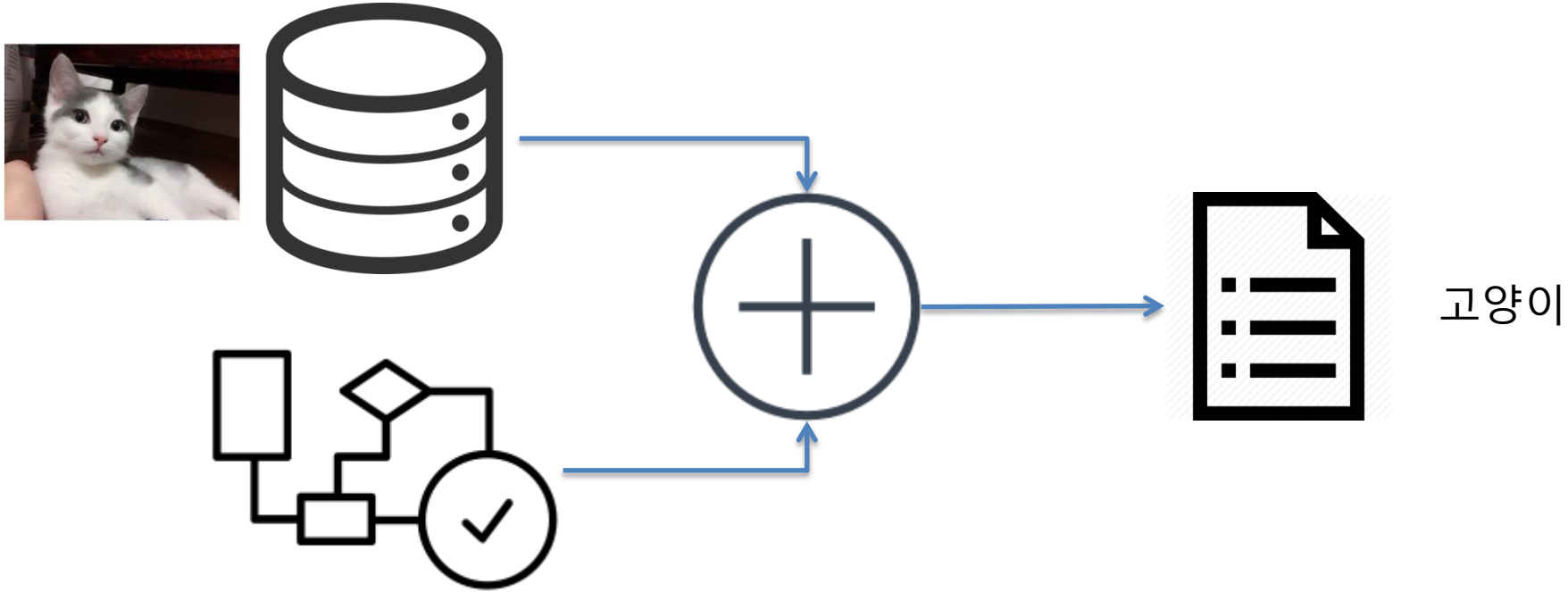
Machine Learning

명시적 프로그래밍



Machine Learning

명시적 프로그래밍



Machine Learning

명시적 프로그래밍

귀, 눈, 코,
발바닥

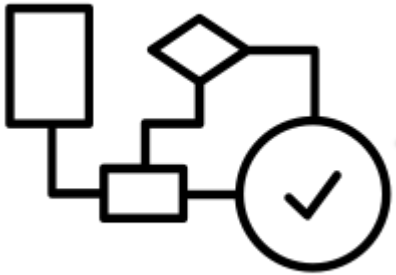
고양이

Machine Learning

명시적 프로그래밍

고양이(?)

귀, 눈, 코,
발바닥

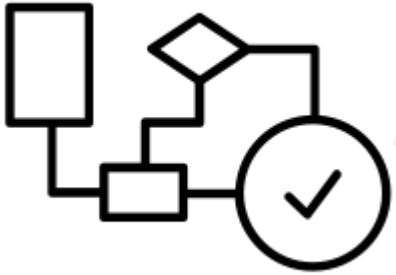


Machine Learning

명시적 프로그래밍

고양이(?)

귀, 눈, 코,
발바닥

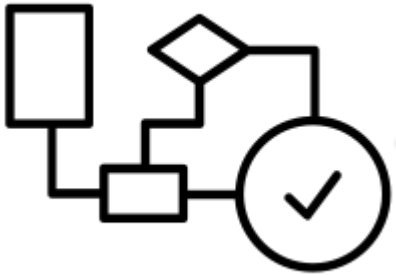


Machine Learning

명시적 프로그래밍

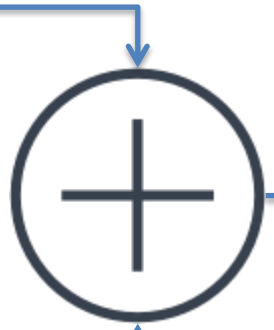
고양이(??)

귀, 눈, 코,
발바닥



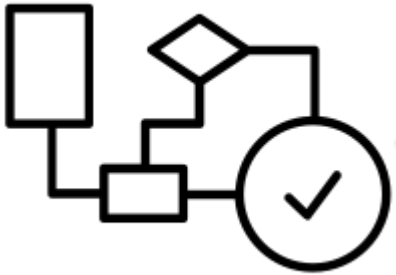
Machine Learning

명시적 프로그래밍



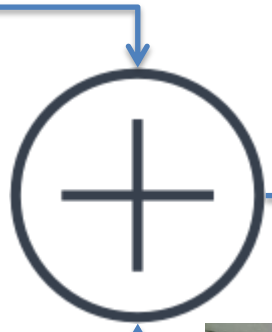
고양이(??)

귀, 눈, 코,
발바닥



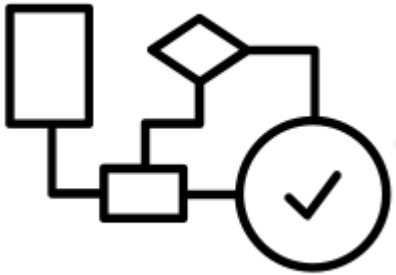
Machine Learning

명시적 프로그래밍



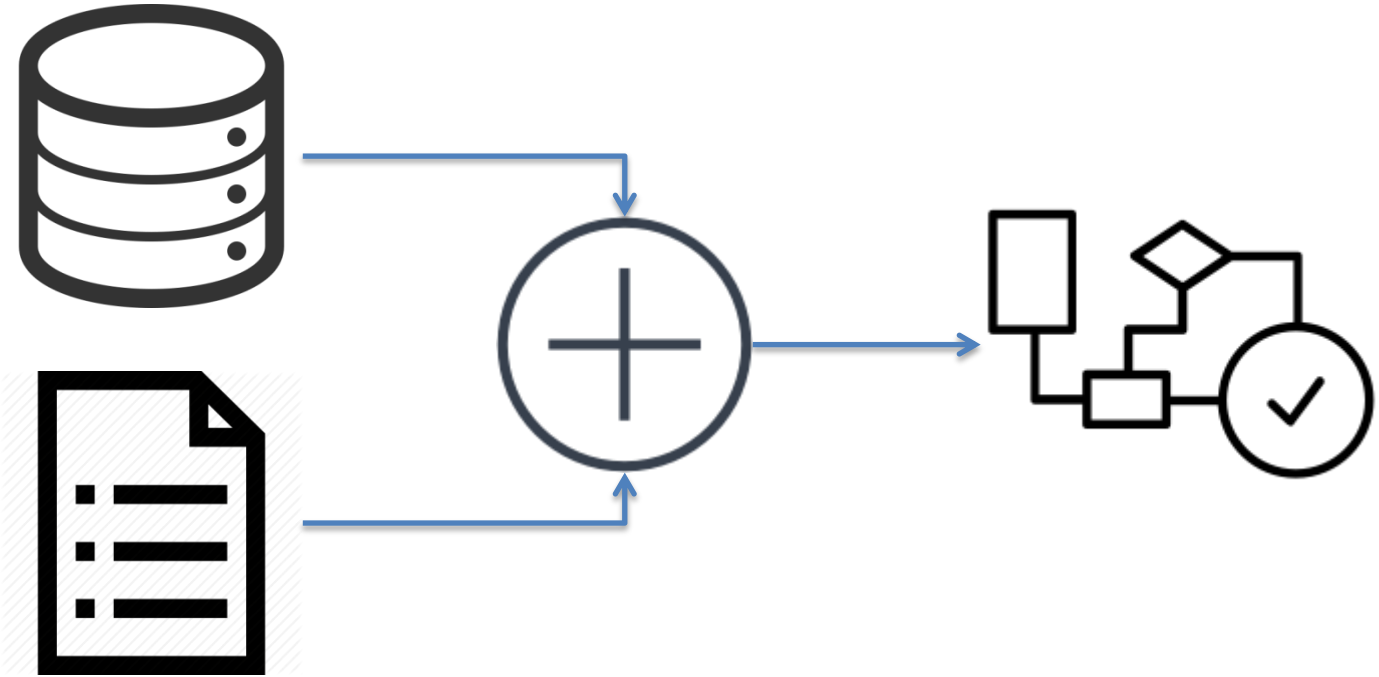
고양이(???)

귀, 눈, 코,
발바닥

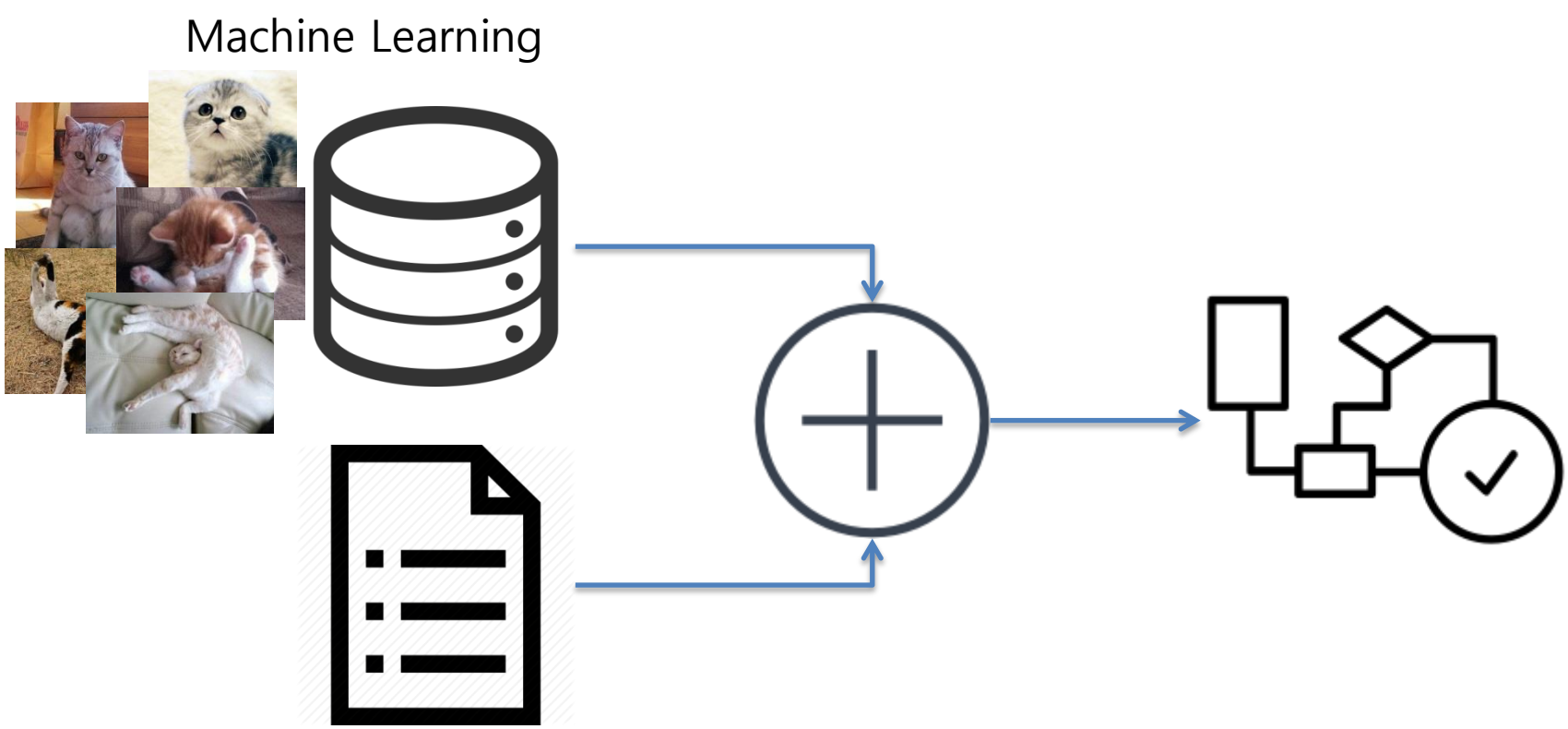


Machine Learning

Machine Learning



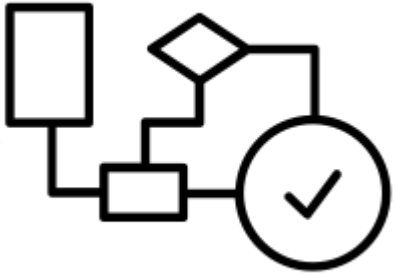
Machine Learning



Machine Learning

Machine Learning

고양이



Machine Learning

Machine Learning

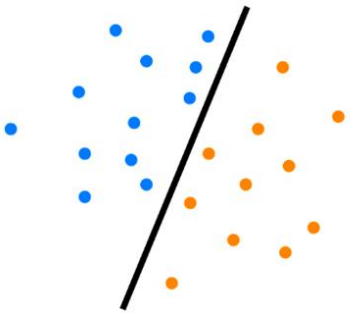
고양이

고양이 판별 함수

Machine Learning

Supervised

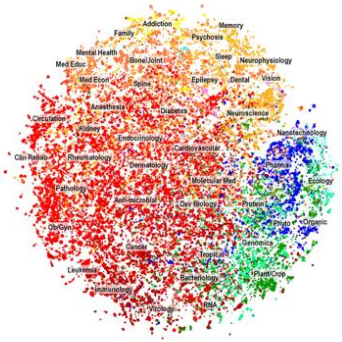
Learning
known
patterns



(위의 고양이판별 예시에 해당)

Unsupervised

Learning
unknown
patterns



Reinforcement

Generating data

Learning patterns

(출처: http://adgefficiency.com)

Deep Learning

Deep learning is the study of artificial neural networks and related machine learning algorithms that contain more than one hidden layer.

Deep Learning

인공 신경망

Deep learning is the study of artificial neural networks and related machine learning algorithms that contain more than one hidden layer.

Deep Learning

인공 신경망

Deep learning is the study of artificial neural networks and related machine learning algorithms that contain more than one hidden layer.

은닉 층

Deep Learning

뜨게 된 이유

Deep Learning

뜨게 된 이유

(출처:luluafrikani.wordpress.com)

Deep Learning

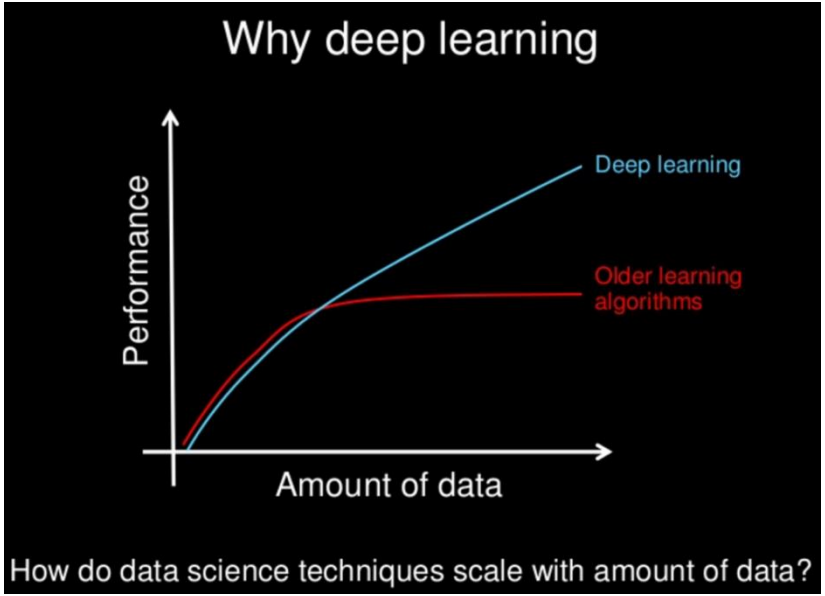
뜨게 된 이유

(출처:luluafrikani.wordpress.com)

(출처:http://wccftech.com)

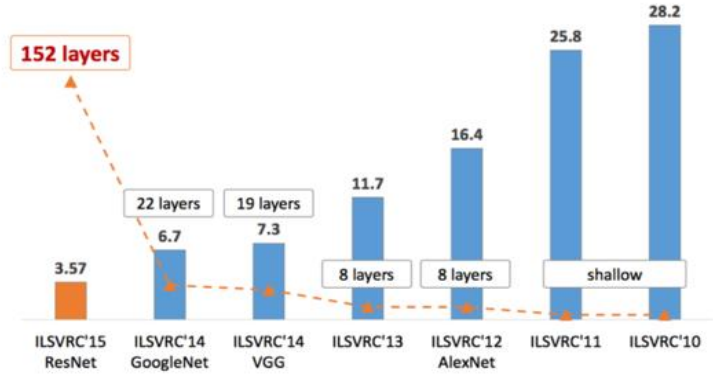
Deep Learning

공부해야 하는 이유



(출처: Andrew Ng)

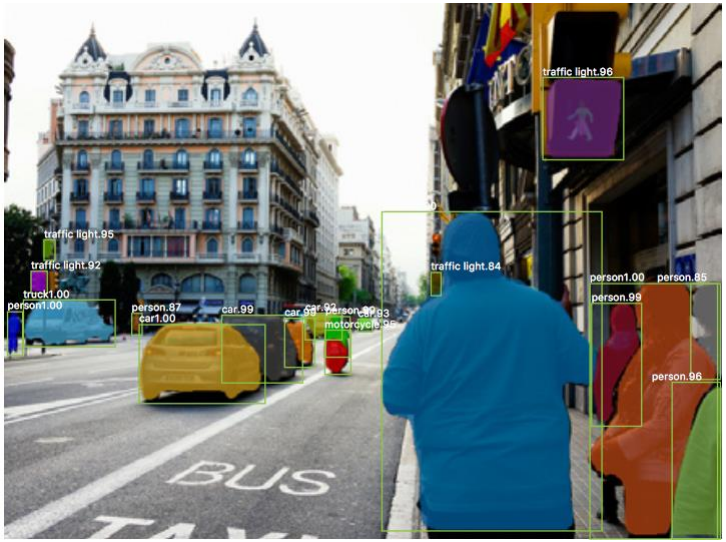
AlphaGo



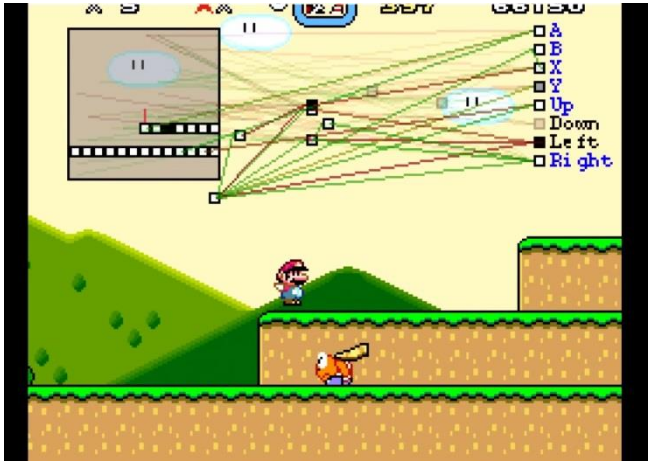
(출처: Kaming He)

Deep Learning

활용 분야



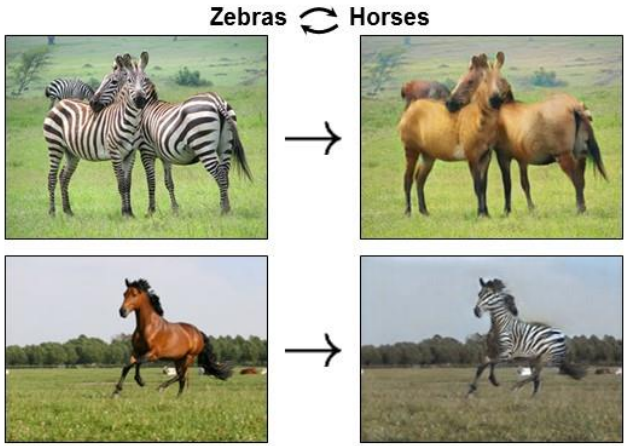
(출처: Mask R-CNN)



(출처: MarI/O)

Deep Learning

활용 분야



(출처: CycleGAN)

(출처: Deep Photo Style Transfer)

PyTorch

Facebook AI Research

Caffe2

PYTORCH

Yann LeCun

DIRECTOR OF AI RESEARCH

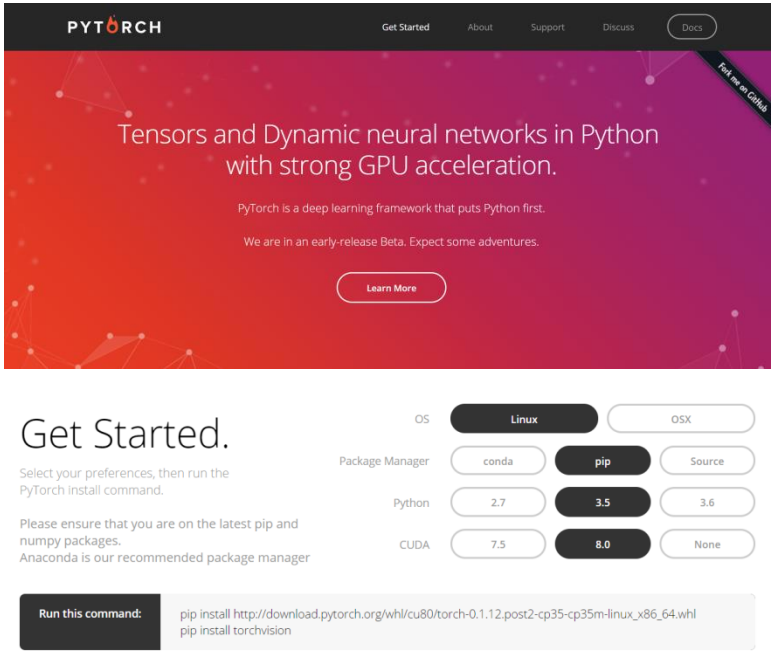
Facebook AI Research (FAIR)

Soumith Chintala

RESEARCH ENGINEER

Facebook AI Research (FAIR)

PyTorch



<파이토치의 특징>

- Python first
- GPU acceleration
- Linux/osx

PyTorch

vs

PYTORCH

In [4]:
```
import numpy as np
from datetime import datetime
start = datetime.now()

np.random.seed(0)

N,D = 3,4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N,D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_y = grad_a * y
grad_x = grad_a * x

print(grad_x)
print(grad_y)
print(grad_z)
print(datetime.now()-start)
```

```
[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.003751
```

In [6]:
```
import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```

PyTorch

vs

PYTORCH

Gradient Calculation

In [4]:
```
import numpy as np
from datetime import datetime
start = datetime.now()

np.random.seed(0)

N,D = 3,4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N,D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_y = grad_a * y
grad_x = grad_a * x

print(grad_x)
print(grad_y)
print(grad_z)
print(datetime.now()-start)
```

```
[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.003751
```

In [6]:
```
import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))
print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```

PyTorch

vs

PYTORCH

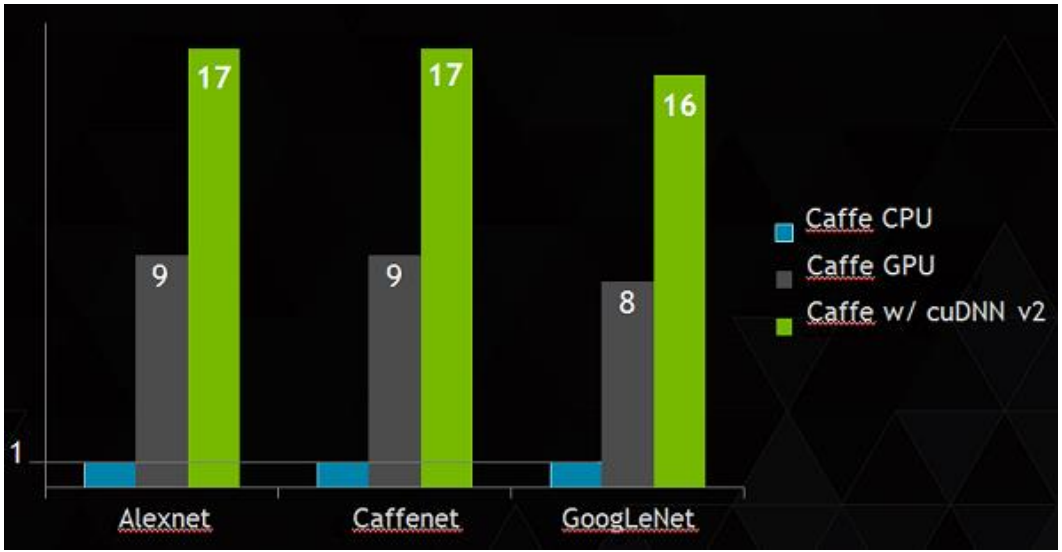
```
In [4]: import numpy as np
from datetime import datetime
start = datetime.now()

np.random.seed(0)

N,D = 3,4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N,D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_y = grad_a * y
grad_x = grad_a * x

print(grad_x)
print(grad_y)
print(grad_z)
print(datetime.now()-start)

[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.003751
```

Gradient Calculation

```
In [6]: import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))
print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)

Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```

걸리는 시간은 비슷(?)

PyTorch



(출처: Nvidia)

<Numpy vs PyTorch>

- 연산량이 많을수록 성능 차이가 커짐
- Automatic Gradient calculation!!
- 모델 자체에 집중할 수 있음

PyTorch

TensorFlow ™ vs PYTORCH

In [5]:
```
import tensorflow as tf
import numpy as np
from datetime import datetime
start = datetime.now()

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c,[x,y,z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N,D),
        y: np.random.randn(N,D),
        z: np.random.randn(N,D)
    }
    out = sess.run([c,grad_x,grad_y,grad_z],feed_dict = values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out

print(grad_x_val)
print(grad_y_val)
print(grad_z_val)
print(datetime.now()-start)
```

```
[[-1.6138978  -0.21274029 -0.89546657  0.38690251]
 [-0.51080513 -1.18063223 -0.02818223  0.42833188]
 [ 0.06651722  0.30247191 -0.63432211 -0.36274117]]
[[ 1.23029065  1.20237982 -0.38732681 -0.30230275]
 [-1.04855299 -1.42001796 -1.70627022  1.95077538]
 [-0.5096522  -0.43807429 -1.25279534  0.77749038]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.046684
```

In [6]:
```
import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```

PyTorch

TensorFlow™ vs PYTORCH

In [5]:
```
import tensorflow as tf
import numpy as np
from datetime import datetime
start = datetime.now()

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c,[x,y,z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N,D),
        y: np.random.randn(N,D),
        z: np.random.randn(N,D)
    }
    out = sess.run([c,grad_x,grad_y,grad_z],feed_dict = values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out

print(grad_x_val)
print(grad_y_val)
print(grad_z_val)
print(datetime.now()-start)
```

```
[[-1.6138978  -0.21274029 -0.89546657  0.38690251]
 [-0.51080513 -1.18063223 -0.02818223  0.42833188]
 [ 0.06651722  0.30247191 -0.63432211 -0.36274117]]
[[ 1.23029065  1.20237982 -0.38732681 -0.30230275]
 [-1.04855299 -1.42001796 -1.70627022  1.95077538]
 [-0.5096522  -0.43807429 -1.25279534  0.77749038]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.046684
```

Define and Run

In [6]:
```
import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```

PyTorch

TensorFlow™ vs PYTORCH

In [5]:
```
import tensorflow as tf
import numpy as np
from datetime import datetime
start = datetime.now()

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c,[x,y,z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N,D),
        y: np.random.randn(N,D),
        z: np.random.randn(N,D)
    }
    out = sess.run([c,grad_x,grad_y,grad_z],feed_dict = values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out

print(grad_x_val)
print(grad_y_val)
print(grad_z_val)
print(datetime.now()-start)
```

```
[[-1.6138978  -0.21274029 -0.89546657  0.38690251]
 [-0.51080513 -1.18063223 -0.02818223  0.42833188]
 [ 0.06651722  0.30247191 -0.63432211 -0.36274117]]
[[ 1.23029065  1.20237982 -0.38732681 -0.30230275]
 [-1.04855299 -1.42001796 -1.70627022  1.95077538]
 [-0.5096522  -0.43807429 -1.25279534  0.77749038]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.046684
```

Define and Run

Define by Run

In [6]:
```
import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```

PyTorch

TensorFlow™ vs PYTORCH

In [5]:
```
import tensorflow as tf
import numpy as np
from datetime import datetime
start = datetime.now()

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c,[x,y,z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N,D),
        y: np.random.randn(N,D),
        z: np.random.randn(N,D)
    }
    out = sess.run([c,grad_x,grad_y,grad_z],feed_dict = values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out

print(grad_x_val)
print(grad_y_val)
print(grad_z_val)
print(datetime.now()-start)
```

```
[[-1.6138978  -0.21274029 -0.89546657  0.38690251]
 [-0.51080513 -1.18063223 -0.02818223  0.42833188]
 [ 0.06651722  0.30247191 -0.63432211 -0.36274117]]
[[ 1.23029065  1.20237982 -0.38732681 -0.30230275]
 [-1.04855299 -1.42001796 -1.70627022  1.95077538]
 [-0.5096522  -0.43807429 -1.25279534  0.77749038]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.046684
```

Define and Run

Define by Run

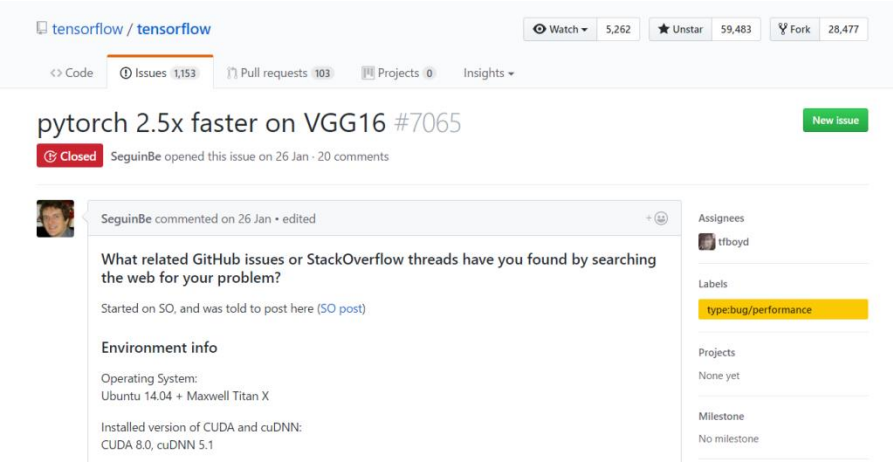
In [6]:
```
import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(),requires_grad=True)
y = Variable(torch.randn(N,D).cuda(),requires_grad=True)
z = Variable(torch.randn(N,D).cuda(),requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward(gradient=torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

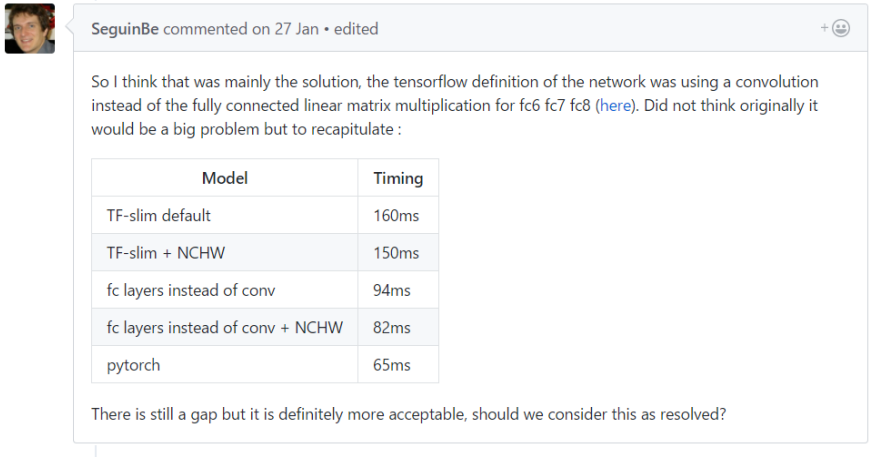
```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 0.1152  1.1809  1.4522  1.9417
 1.0845  0.1587 -1.6526  0.4031
 1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
 1  1  1  1
 1  1  1  1
 1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]
0:00:00.003434
```

TF가 10배 이상 느리다??

PyTorch

TensorFlow ™ vs PYTORCH





"좀 자극적인 제목 + 실험 환경 확인불가"이지만 파이토치가 빠르긴 빠르다.

PyTorch

vs

- Define and Run
- Static graph
- 사용자 많음
- 자체 운영 포럼 없음
- TF-KR

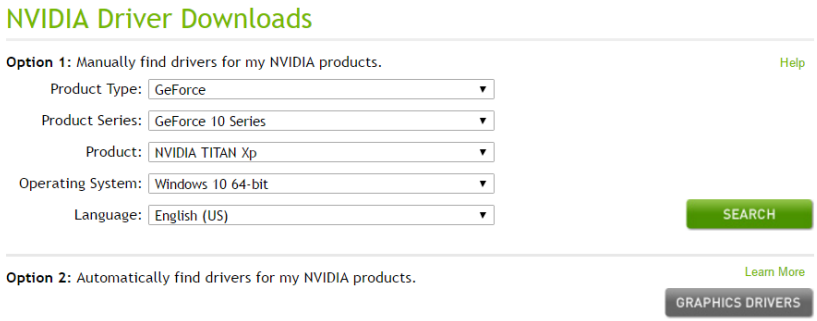
- Define by Run
- Dynamic graph
- 늘어나는 추세
- 자체적 포럼 운영
- PyTorch-KR

PyTorch

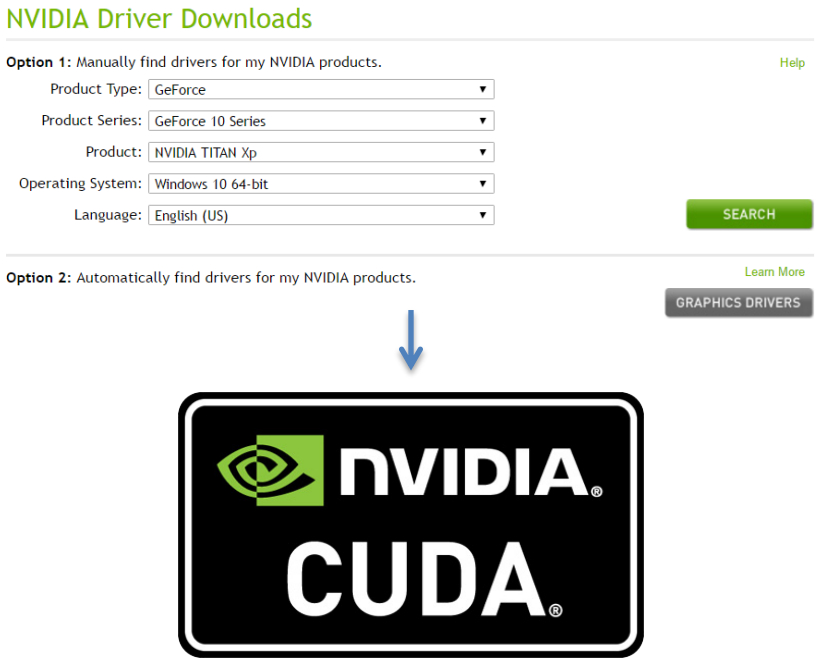
Stanford CS231n 강의자 & Tesla Researcher

Installation

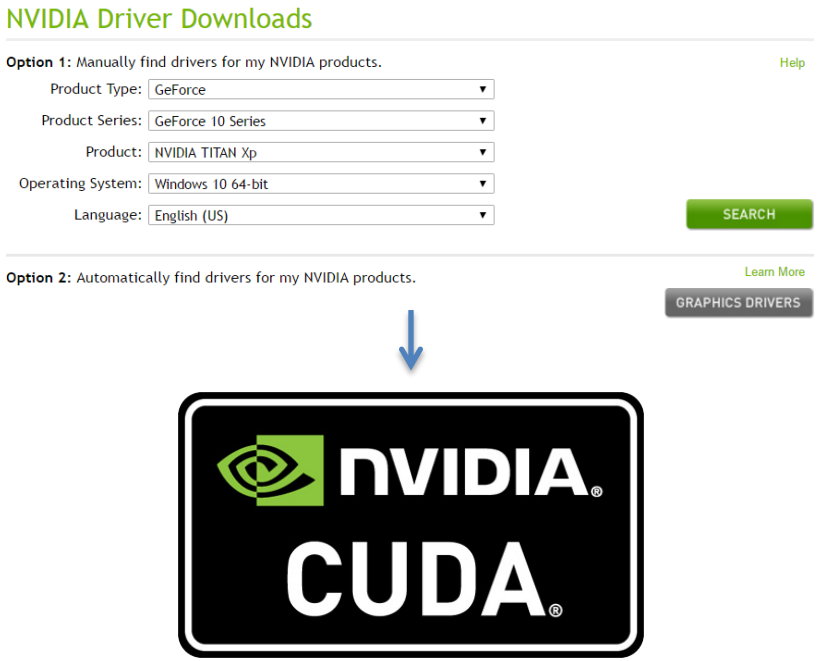
Installation



Installation

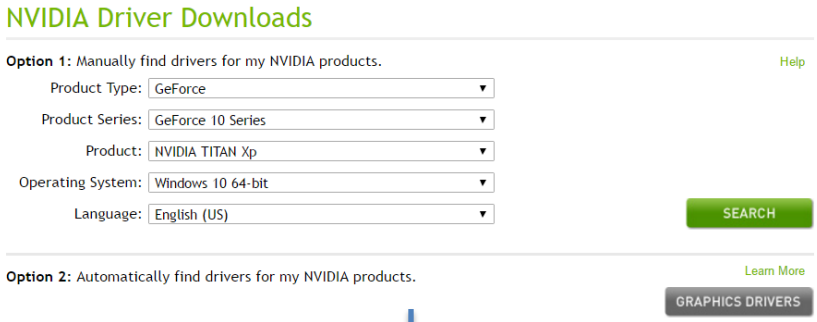


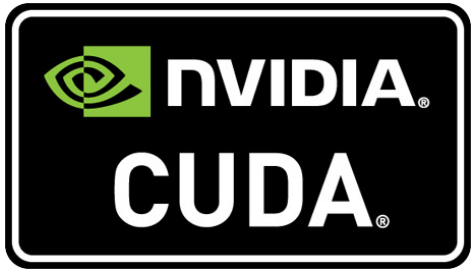
Installation



병렬컴퓨팅 플랫폼이자 API. 일반적인 GPU를 그래픽스만이 아닌 범용으로 쓸 수 있게 해준다.
GPGPU (General-Purpose computing on Graphics Processing Units)

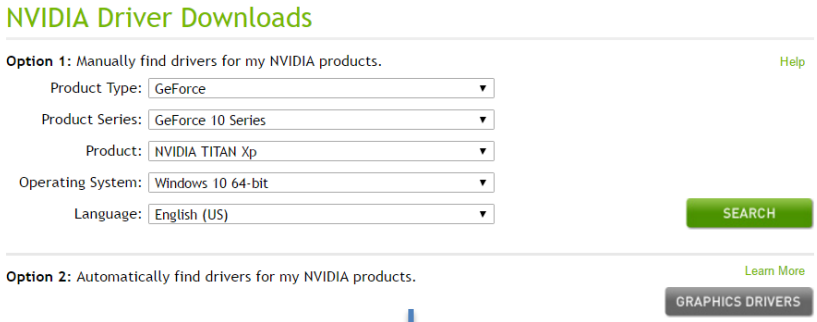
Installation

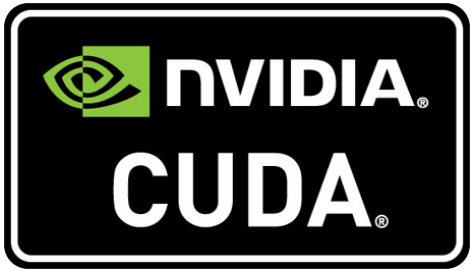




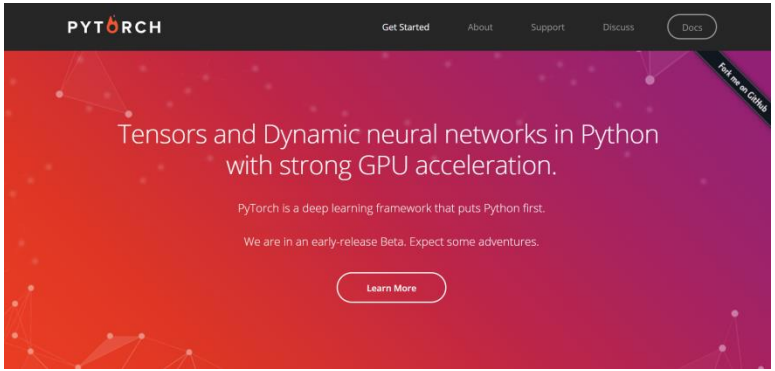
딥러닝에 특화된 NVIDIA GPU 라이브러리. CuDNN을 사용하면 CUDA만을 썼을때보다 2~3배 빨라진다고 함

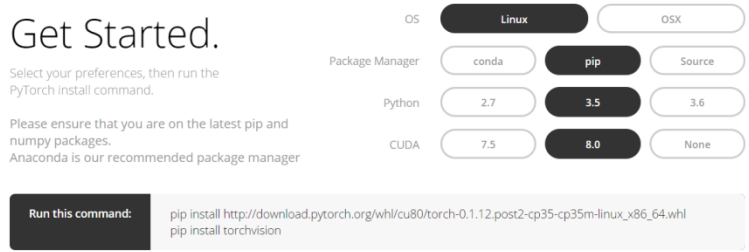
Installation





Installation





파이썬, 엔비디아 그래픽 드라이버, CUDA, CUDNN 모두 설치한 후에 pip 또는 conda로 설치하면 드디어 쓸 수 있습니다.

노트북에도 깔고 회사 서버에도 깔고,
친구 깔아주고 서버 터지면 다시 깔고
총 50~60번은 설치를 시도했던 것 같습니다.

각자 노트북 사양에 맞게 깔아야 하는 거라
엄청 세세한 부분은 생략했습니다.

(참고링크: https://alliseesolutions.wordpress.com/2016/09/08/install-gpu-tensorflow-from-sources-w-ubuntu-16-04-and-cuda-8-0/)

Packages

Package	Description
torch	a Tensor library like NumPy, with strong GPU support
torch.autograd	a tape based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.nn	a neural networks library deeply integrated with autograd designed for maximum flexibility
torch.optim	an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.
torch.multiprocessing	python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and hogwild training.
torch.utils	DataLoader, Trainer and other utility functions for convenience
torch.legacy(.nn/.optim)	legacy code that has been ported over from torch for backward compatibility reasons

+

torchvision

Packages

Package	Description
torch	a Tensor library like NumPy, with strong GPU support

- Tensor
- Creation Ops
- Indexing, Slicing, Joining, Mutating Ops
- Random sampling
- Serialization
- Parallelism
- Math Operations

Packages

Package	Description
torch.autograd	a tape based automatic differentiation library that supports all differentiable Tensor operations in torch

- Variable
- Function

Packages

Package	Description
torch.nn	a neural networks library deeply integrated with autograd designed for maximum flexibility

- Parameters
- Containers
- Convolution Layers
- Pooling Layers
- Non-linear Activations
- Normalization Layers
- Recurrent Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- Multi-GPU Layers
- Utilities

Packages

Package	Description
torch.optim	an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.

- How to use an optimizer
- Algorithms

Packages

Package	Description
torch.multiprocessing	python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and hogwild training.

- Strategy Management
- Sharing CUDA tensors
- Sharing strategies

Packages

Package	Description
torch.utils	DataLoader, Trainer and other utility functions for convenience

- utils.ffi.create_extension
- utils.data
- Utils.model_zoo

Packages

Package	Description
torch.legacy(.nn/.optim)	legacy code that has been ported over from torch for backward compatibility reasons

기존 torch 함수들

Packages

torchvision Reference

- torchvision
- torchvision.datasets
- torchvision.models
- torchvision.transforms
- torchvision.utils

Packages

torchvision Reference

- torchvision
- torchvision.datasets
- torchvision.models
- torchvision.transforms
- torchvision.utils

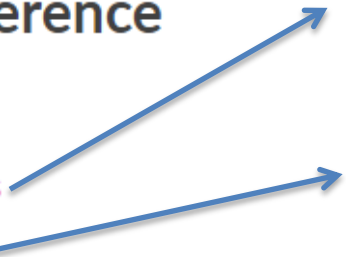
많이 사용되는 데이터들

- ImageFolder(data loader)
- MNIST
- COCO
- LSUN
- Imagenet-12
- CIFAR10 and CIFAR100
- STL10

Packages

torchvision Reference

- torchvision
- torchvision.datasets
- torchvision.models
- torchvision.transforms
- torchvision.utils



많이 사용되는 데이터들

많이 사용되는 모델들

- AlexNet
- VGG
- ResNet
- SqueezNet
- DenseNet

Packages

torchvision Reference

- torchvision
- torchvision.datasets
- torchvision.models
- torchvision.transforms
- torchvision.utils

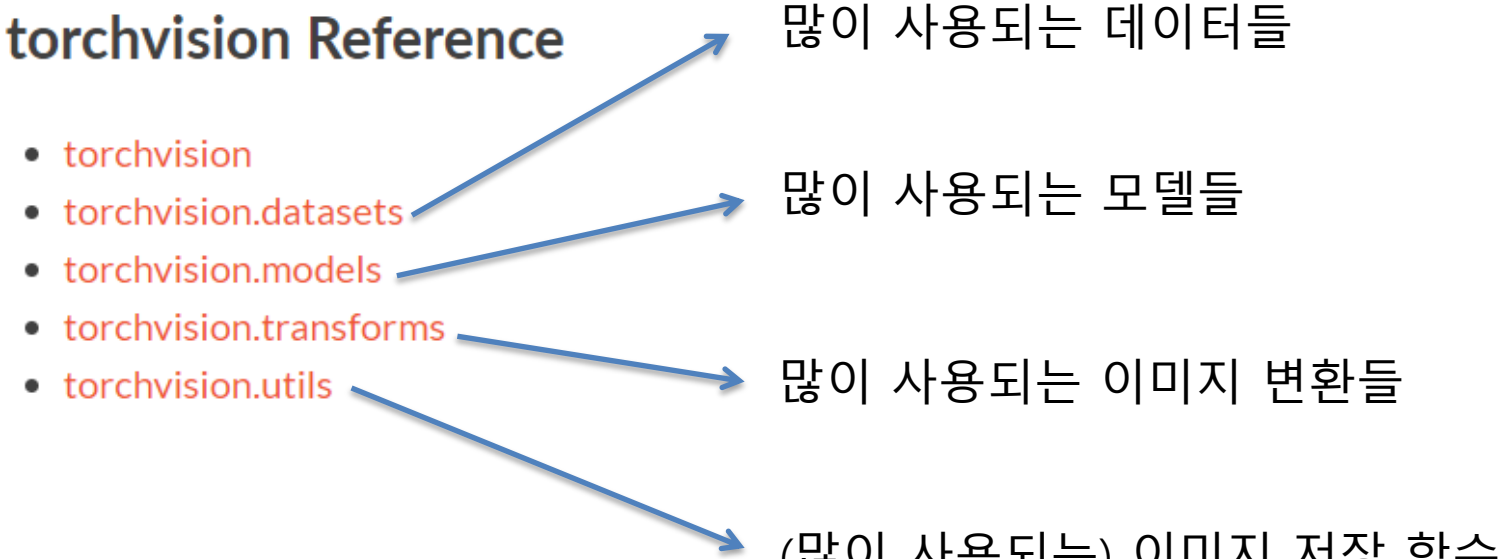
많이 사용되는 데이터들

많이 사용되는 모델들

많이 사용되는 이미지 변환들

- Scale
- Pad
- Crop
- Normalize
- Flip

Packages


torchvision Reference
torchvision
torchvision.datasets
torchvision.models
torchvision.transforms
torchvision.utils
많이 사용되는 데이터들
많이 사용되는 모델들
많이 사용되는 이미지 변환들
(많이 사용되는) 이미지 저장 함수

Q&A