深度学习

– 用 PYTHON 开发你的智能应用

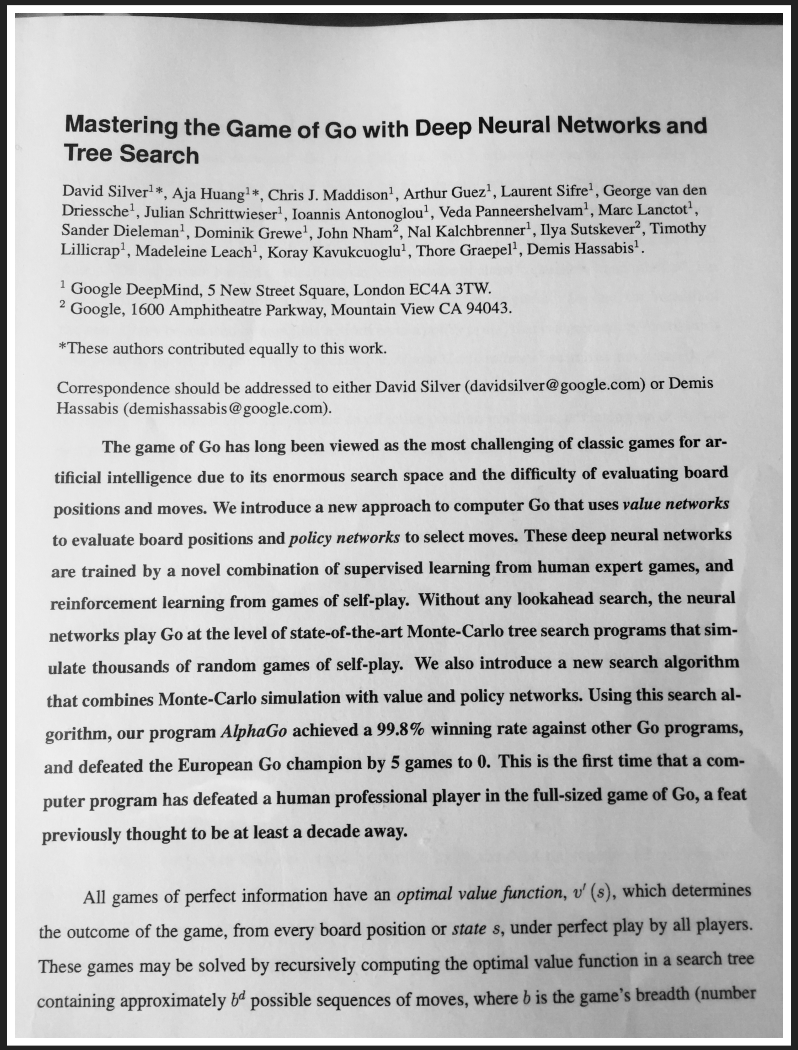
费良宏 / lianghon@amazon.com， *AWS Technical Evangelist*

21 April 2016

关于我

- 工作：Amazon Web Services / Evangelist
  - 7 年 Windows/ Internet/ Cloud @**Microsoft**
  - 3 年 iOS/ Mobile App @**Apple**
  - 1.5 年 Cloud Computing @**AWS**
- 技术关注：
  - **云计算**：架构、大数据、计算优化
  - **机器学习**：深度学习、自然语言处理
  - **语言**：Python、Go、Scala、Lua
  - **Web**：爬虫
- 2016的目标：Web爬虫 + 深度学习 + 自然语言处理 = ？

今年最激动人心的事件?



Mastering the Game of Go with Deep Neural Networks and Tree Search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1], Demis Hassabis[1].

[1] Google DeepMind, 5 New Street Square, London EC4A 3TW.
[2] Google, 1600 Amphitheatre Parkway, Mountain View CA 94043.

*These authors contributed equally to this work.

Correspondence should be addressed to either David Silver (davidsilver@google.com) or Demis Hassabis (demishassabis@google.com).

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence due to its enormous search space and the difficulty of evaluating board positions and moves. We introduce a new approach to computer Go that uses *value networks* to evaluate board positions and *policy networks* to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte-Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte-Carlo simulation with value and policy networks. Using this search algorithm, our program *AlphaGo* achieved a 99.8% winning rate against other Go programs, and defeated the European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

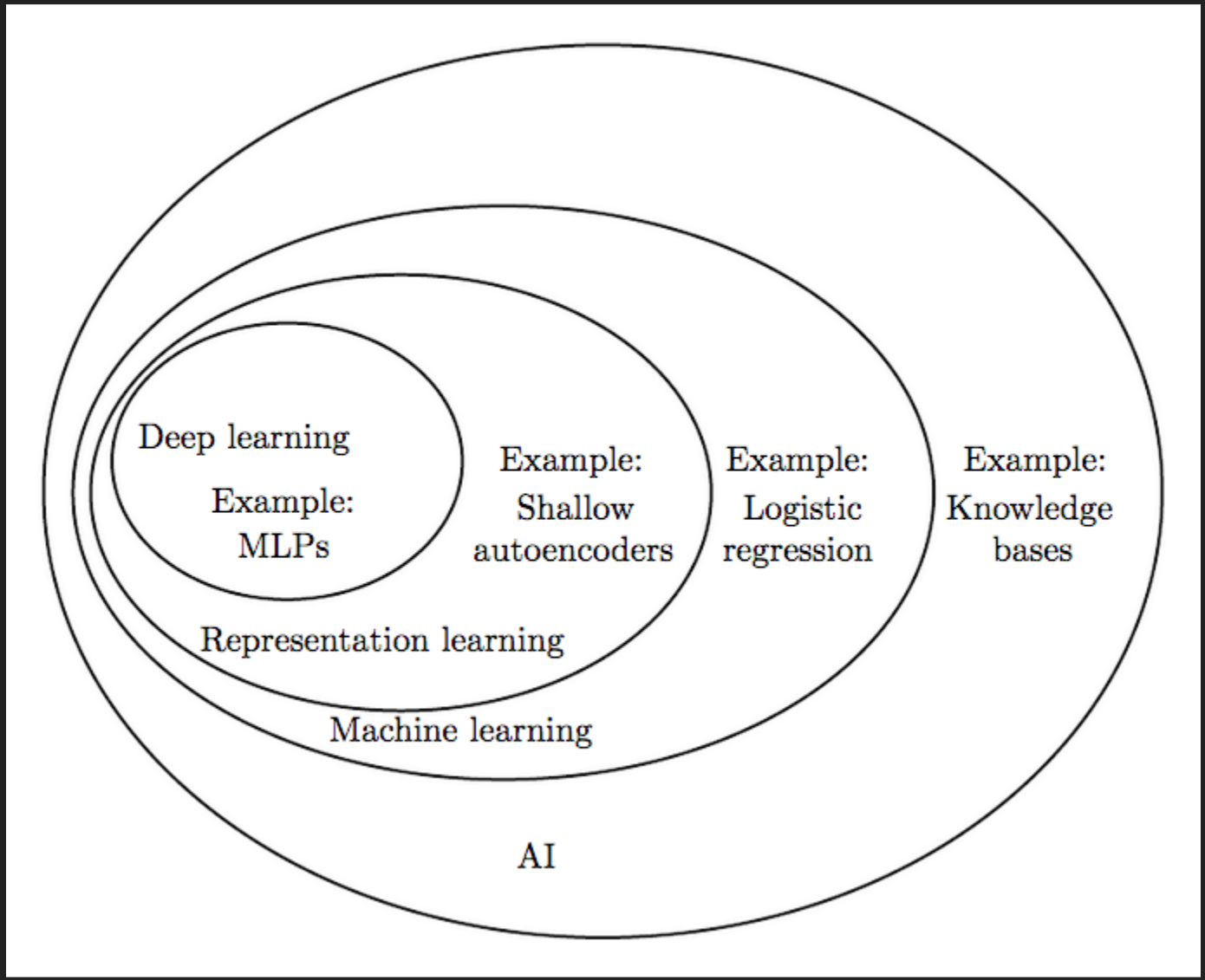
All games of perfect information have an *optimal value function*, $v^*(s)$, which determines the outcome of the game, from every board position or *state* s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number

2016.1.28 “Mastering the game of Go with deep neural networks and tree search”

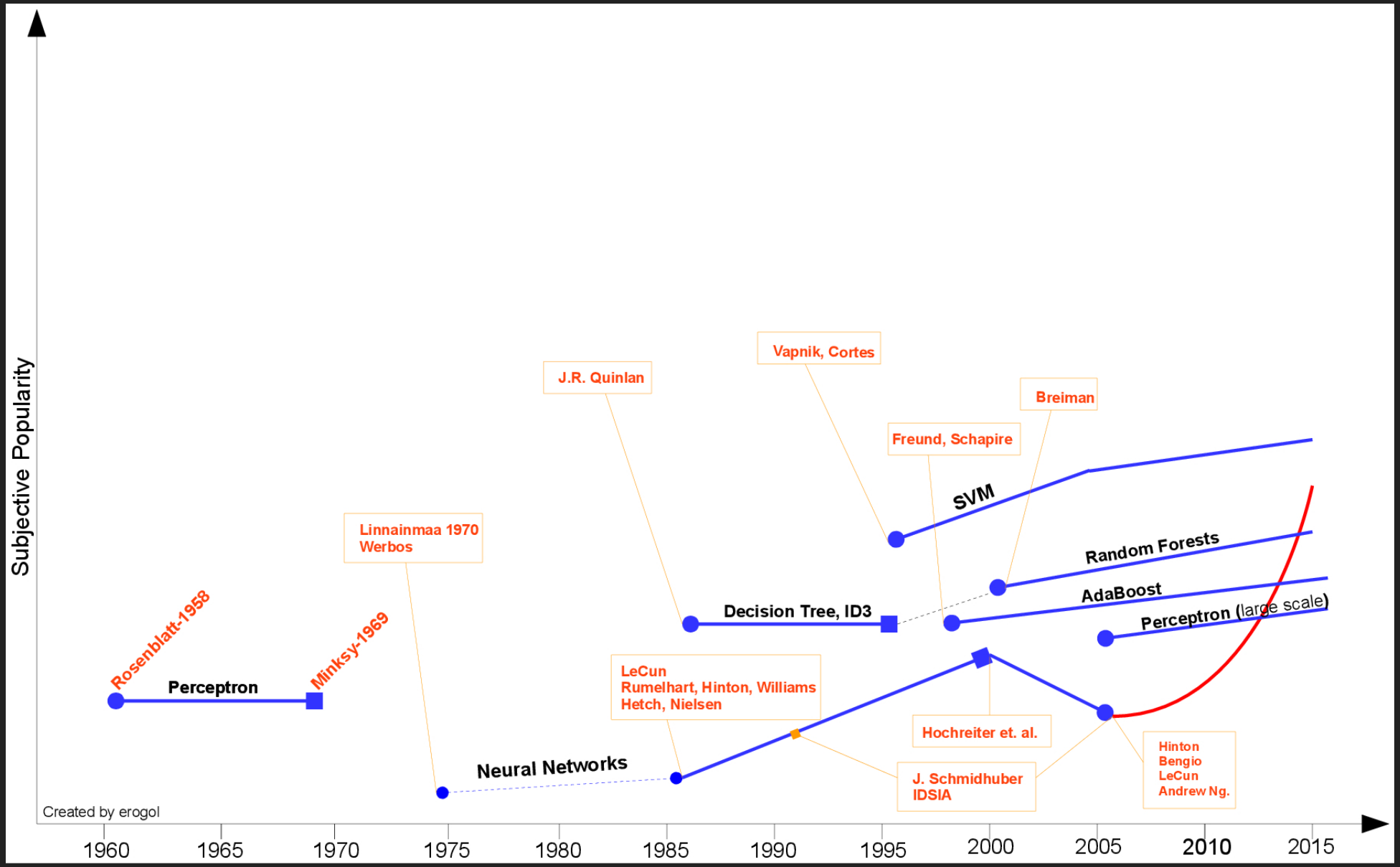
今年最激动人心的事件?

2016年3月Alphago 4:1 击败李世石九段

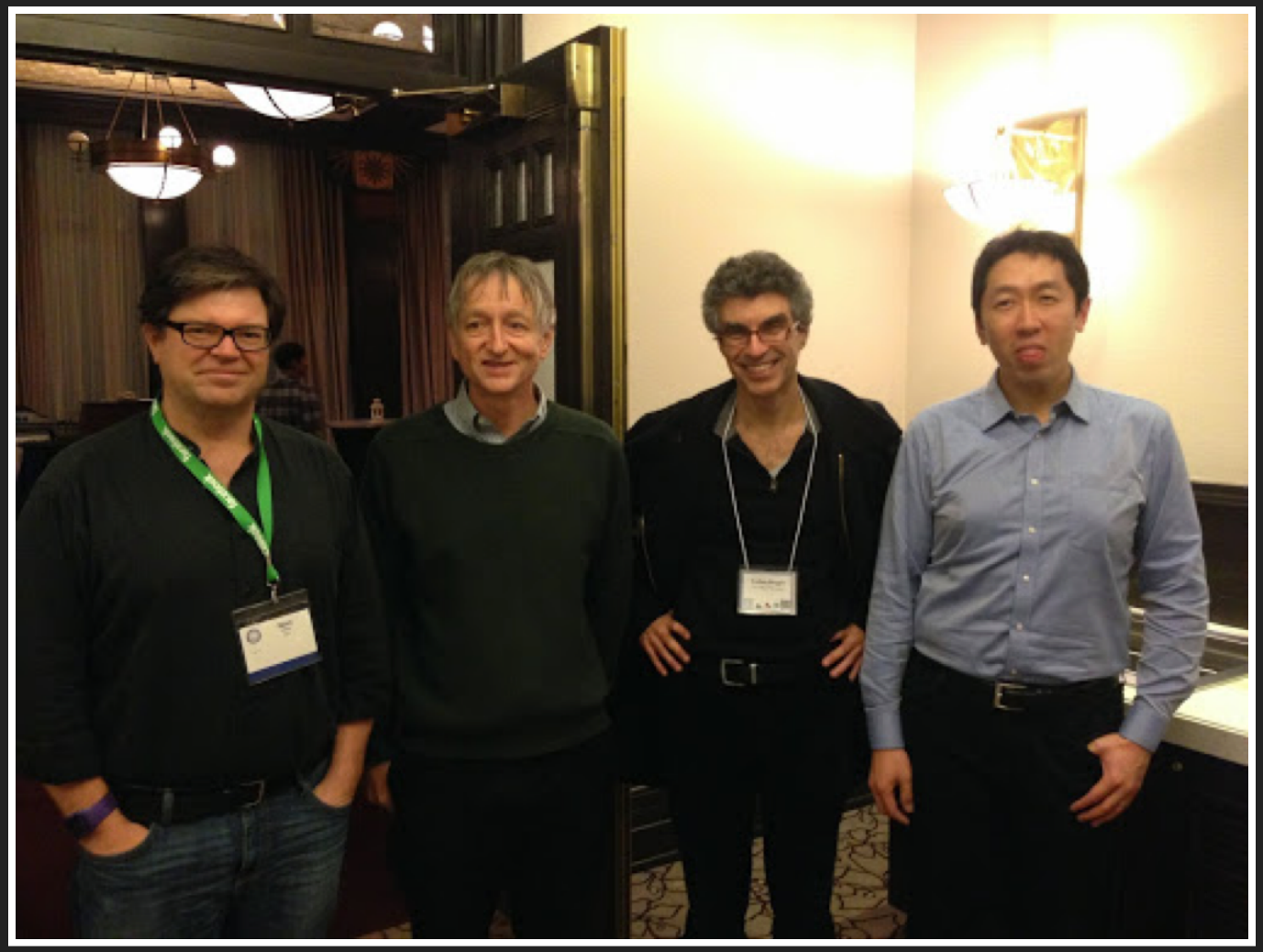
人工智能 VS. 机器学习 VS. 深度学习



人工智能发展的历史


Subjective Popularity
Rosenblatt-1958
Perceptron
Minksy-1969
Linnainmaa 1970
Werbos
J.R. Quinlan
Vapnik, Cortes
Breiman
Freund, Schapire
SVM
Random Forests
AdaBoost
Perceptron (large scale)
Decision Tree, ID3
LeCun
Rumelhart, Hinton, Williams
Hetch, Nielsen
Neural Networks
Hochreiter et. al.
J. Schmidhuber
IDSIA
Hinton
Bengio
LeCun
Andrew Ng.
Created by erogol
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015

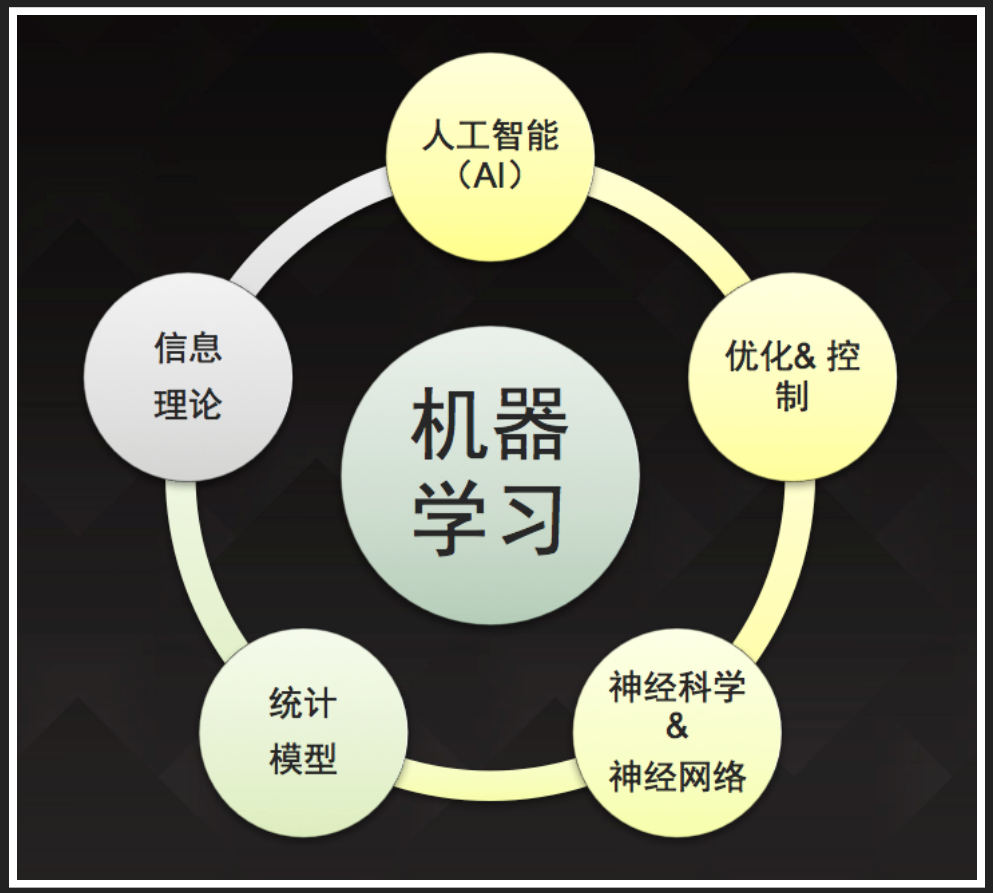
四大宗师



Yann Lecun, Geoff Hinton, Yoshua Bengio, Andrew Ng

机器学习

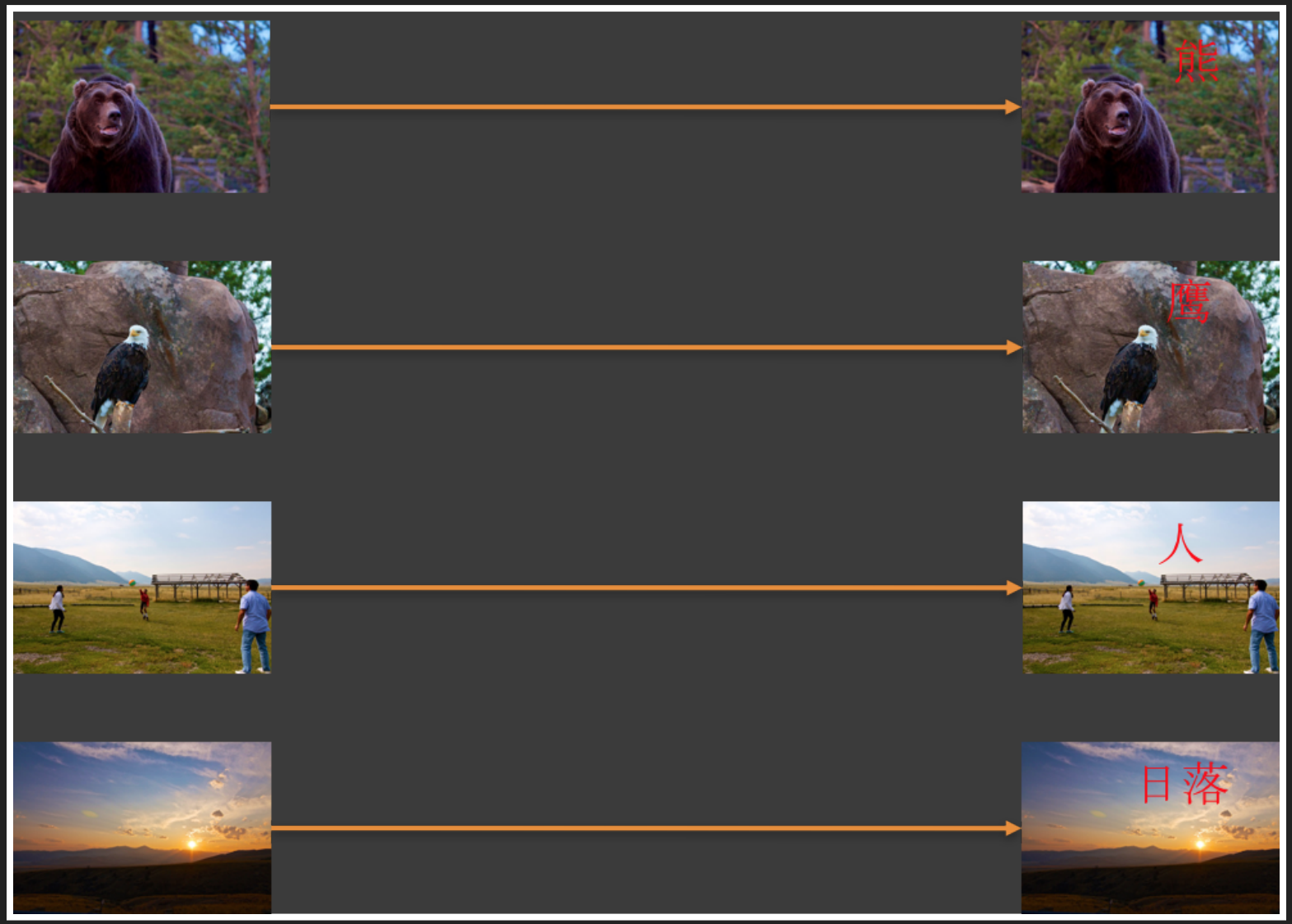
机器学习是一门人工智能的科学。机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法



机器学习

计算机能够分辨出来他／她是谁吗?

机器学习



机器学习

- 基于过去的事实和数据，用来发现趋势和模式
- 机器学习模型提供了对于结果的洞察力，机器学习帮助揭示未来的一个结果的概 率而不仅仅是过去发生的事情
- 历史的数据和统计建模被用于概率进行预测

传统的数据分析旨在回答关于过去的事实，机器学习的目的是回答关于未来事件的可能性的问题!

机器学习的应用场景

个性化 – 提供个性化的电子商务体验

文档聚类 – 按照文档上下文自动分类

欺诈检测 – 发现异常的规律行为，识别和标记欺诈交易

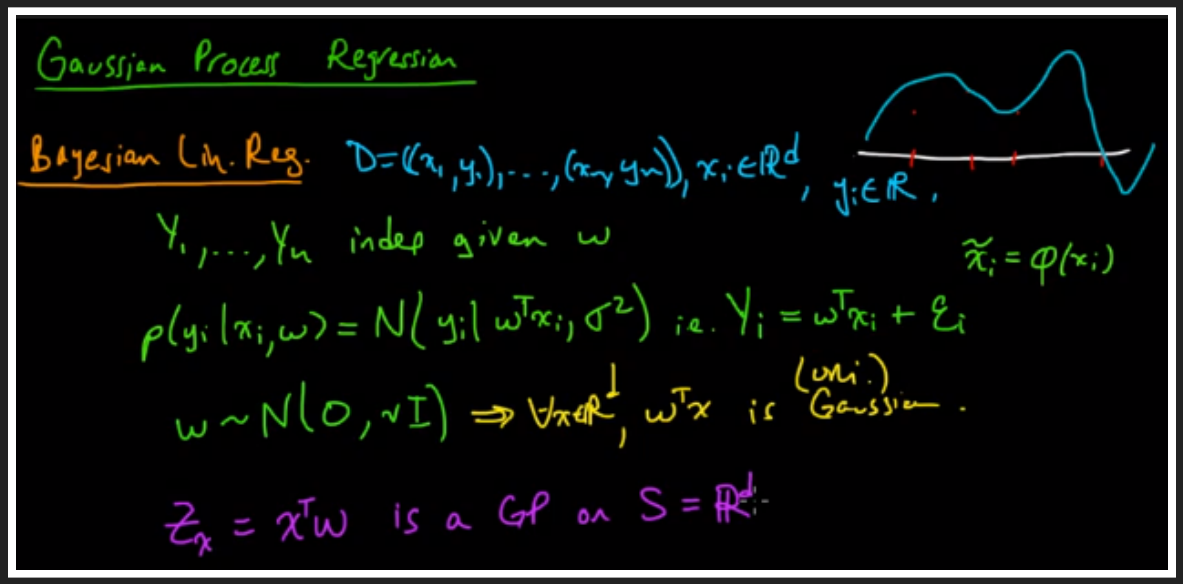
推荐引擎

客户流失预测

...

机器学习 – 学习方式

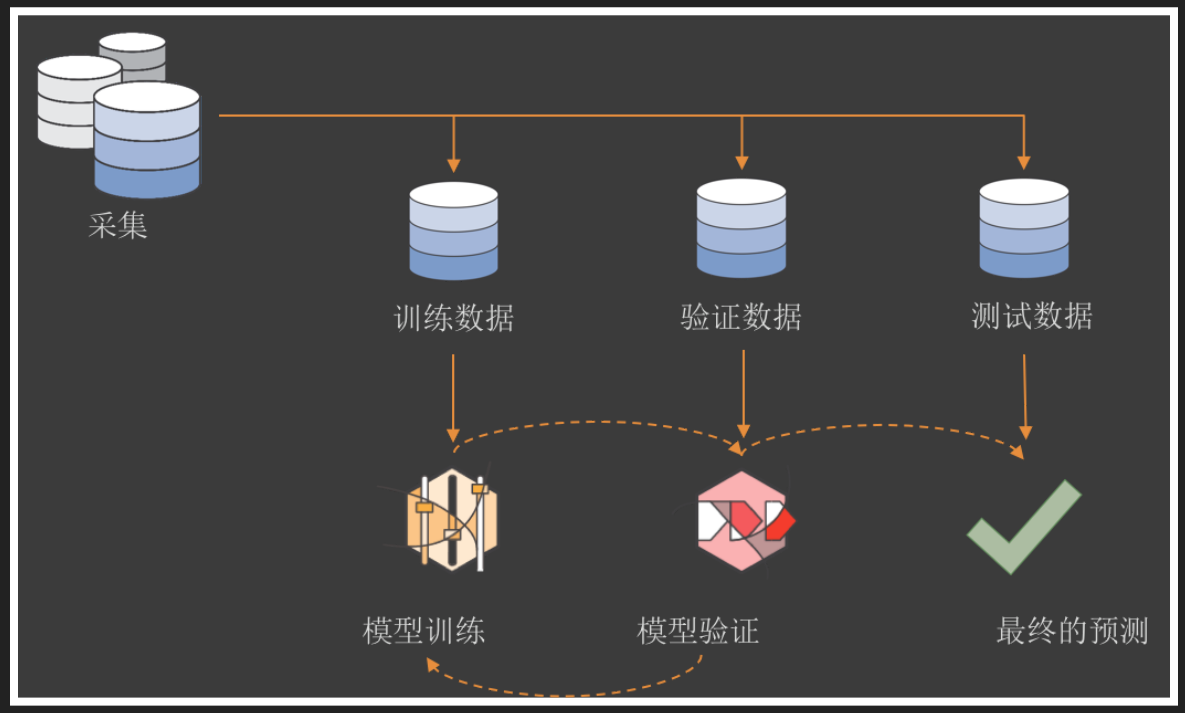
- 监督学习- 人工干预和验证的要求,算法： Logistic Regression， Back Propagation Neural Network 等。例如： 照片分类和标签
- 无监督学习- 无人工干预的要求, 算法: Apriori算法以及k-Means。例如： 对于文档的基于上下文的自动分类
- 半监督学习 - 介于监督学习和无监督学习之间， 算法: Graph Inference 或者Laplacian SVM
- 强化学习- 通过观察来学习做成如何的动作, 算法： Q-Learning以及时间差学习



机器学习 – 方法及流程

- 输入特征选择 – 基于什么进行预测
- 目标 – 预测什么
- 预测功能 – 回归、聚类、降维...

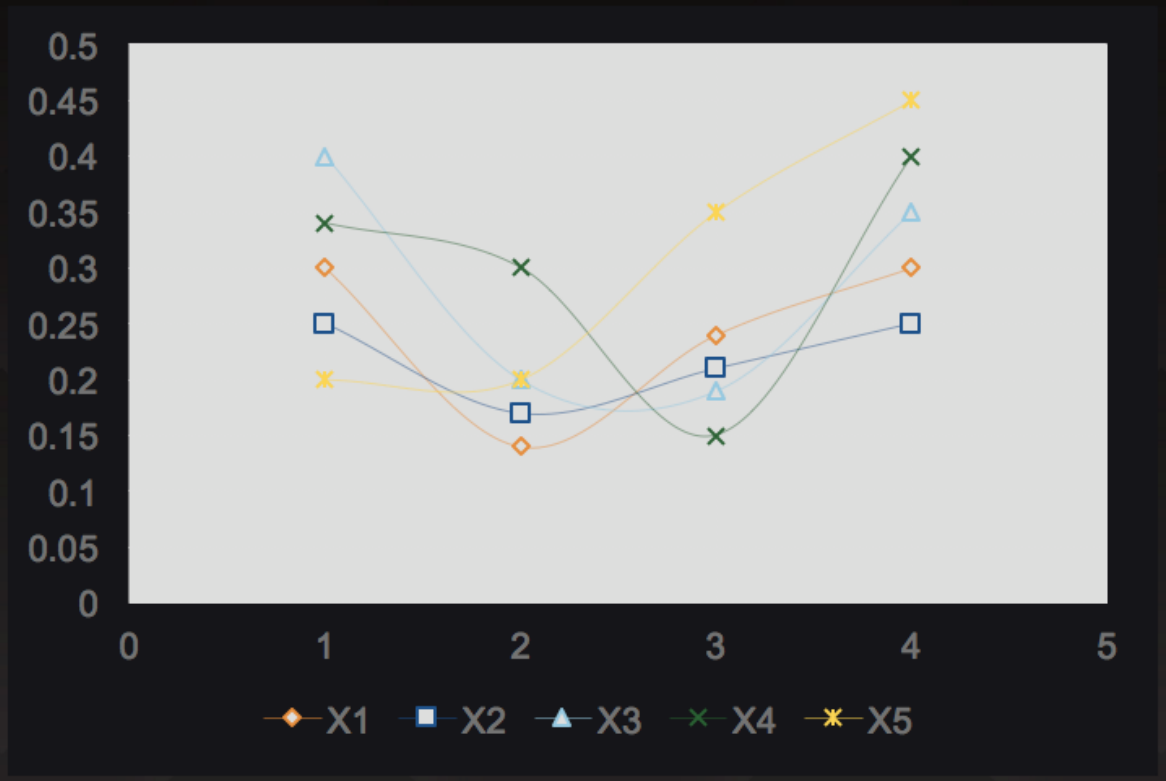
Xn -> F(xn) -> T(x)



机器学习 – 举例

$\chi_n \epsilon F(xn)^{\square}$; Target: y

X_1	X_2	X_3	X_4	X_5	Y
0.3	0.25	0.4	0.34	0.2	1
0.14	0.17	0.2	0.3	0.2	0
0.24	0.21	0.19	0.15	0.35	1
0.3	0.25	0.35	0.4	0.45	1



机器学习 – 举例

- 如何让机器分辨出来他／她是谁？
- 图像分析 –
  输入特征选择 ->面部特征、发型、裙子、身高、手势...

机器学习 – 何时使用

你不需要机器学习，如果 -

- 使用简单的规则和计算，你可以预测答案
- 你能够预先了解到所需要的步骤不需要任何数据驱动的学习

你需要机器学习，如果 -

- 简单的聚类规则是不充分的
- 面对大量的数据集的可伸缩性的问题

机器学习 – 总结

由已知答案的数据开始

明确目标 – 从数据中希望可以预测什么

选择可以被用来预测目标的模式所需要的变量／特性

使用已知目标答案的数据训练机器学习模型

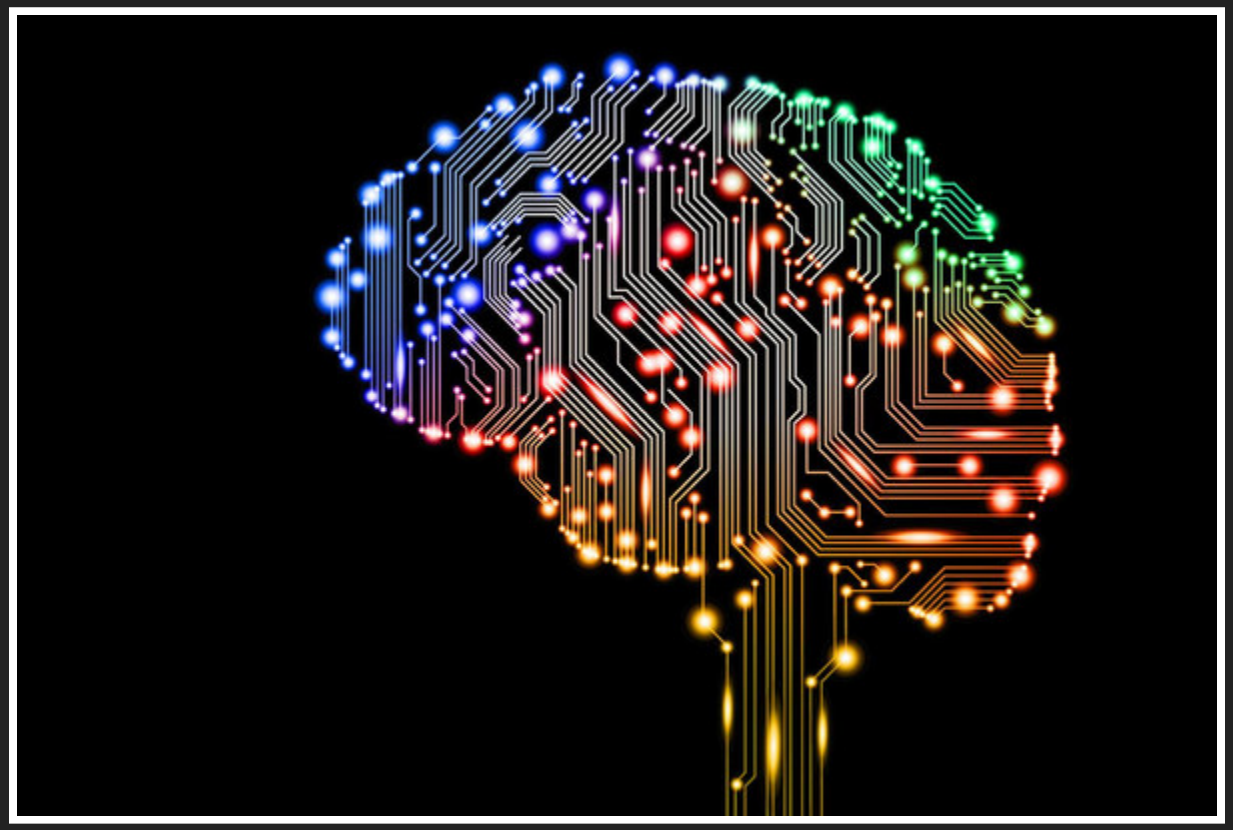
对于未知答案的数据，使用训练过的模型预测目标

评估模型的准确性

提高模型精度

什么是深度学习?

"深度学习是机器学习的一个分支，是一组在多个层次上学习的算法，分别对应不同级别的抽象"

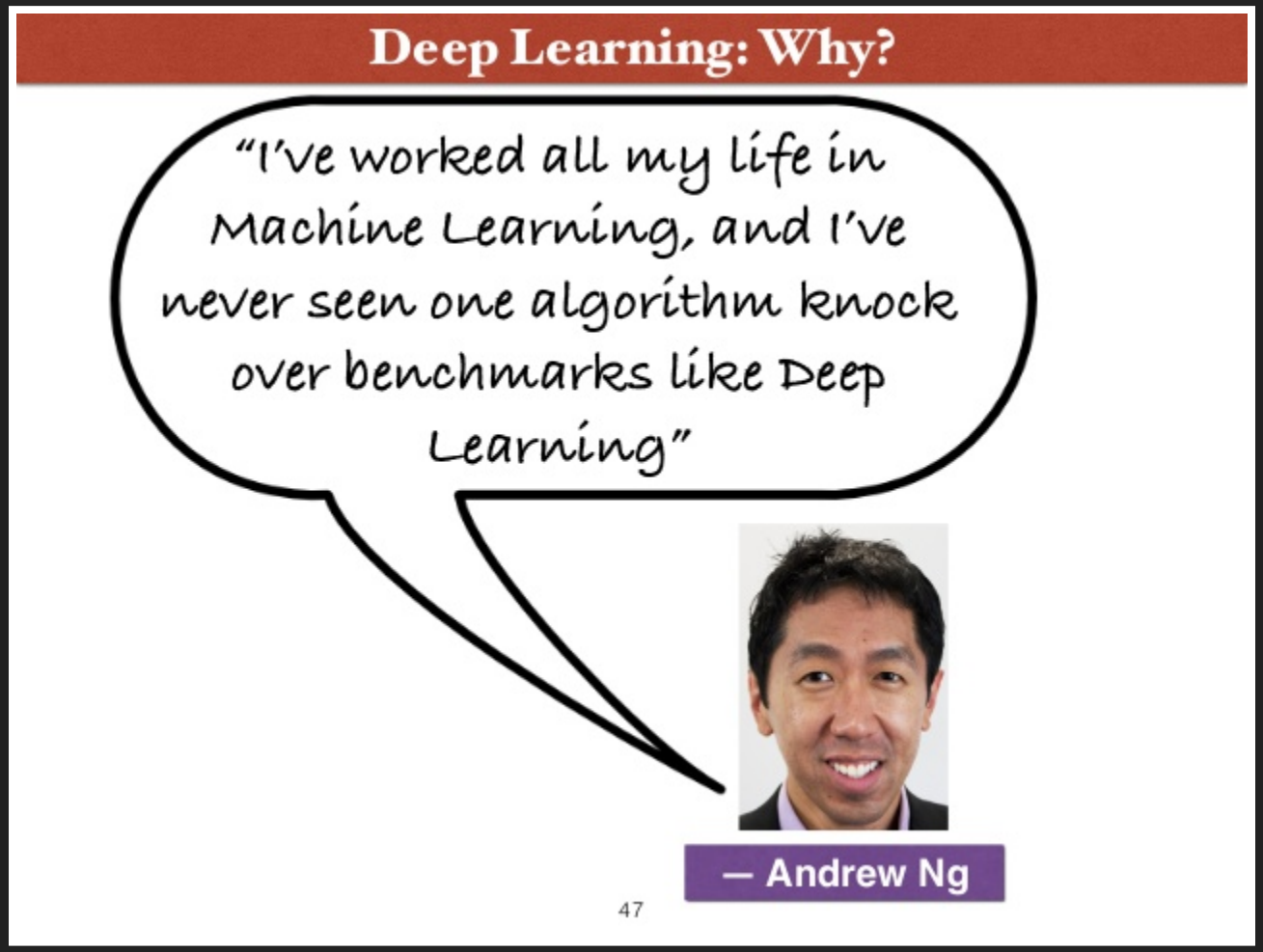


深度学习 VS. 机器学习

- ML 的算法包括监督学习和无监督学习
- 适用非线性处理单元的多层次的特征提取和转换
- 基于对多个层的特征或者表象的学习，形成一个由低级到高级的层次结构特征

传统的机器学习关注于特征工程,深度学习关注于端到端的基于原始数据的学习

为什么需要深度学习?


Deep Learning: Why?
"I've worked all my life in Machine Learning, and I've never seen one algorithm knock over benchmarks like Deep Learning"
— Andrew Ng
47

深度学习- 举例

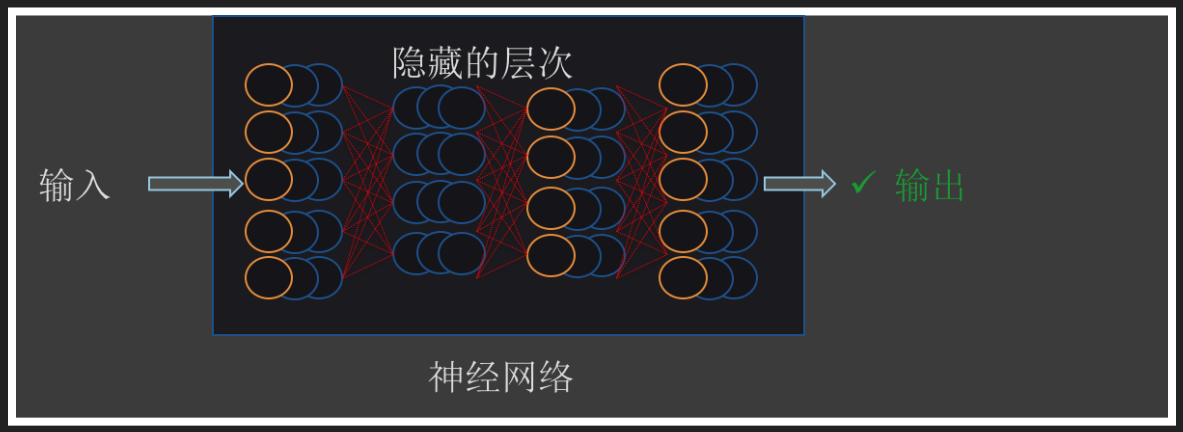

熊
目标: 熊
位置: 黄石公园
动作: 觅食
鹰
目标: 鹰
位置: 沃库拉泉，佛罗里达
动作: 休息
人
目标: 多个 – 人、球
位置: 蒙大拿
动作: 玩耍
日落
目标: ?
位置: 蒙大拿
动作: ?

深度学习 – 神经网络

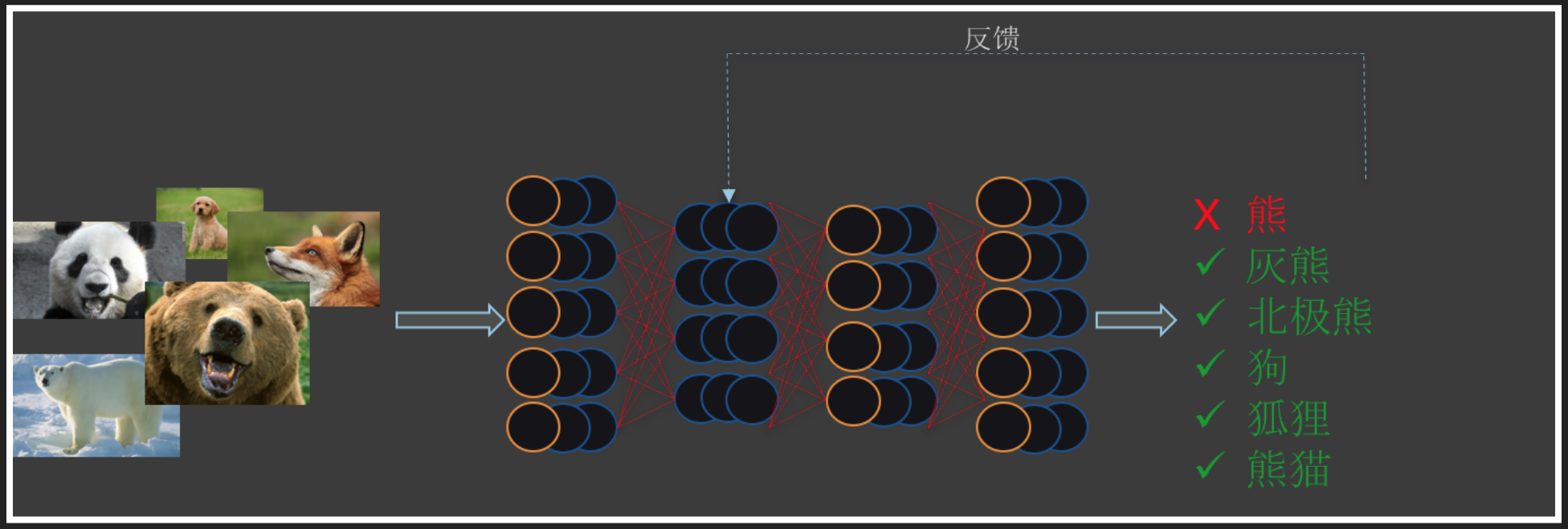
是一种模仿生物神经网络(例如大脑)的结构和功能的计算模型

是一种非线性统计性数据建模工具，对输入和输出间复杂的关系进行建模

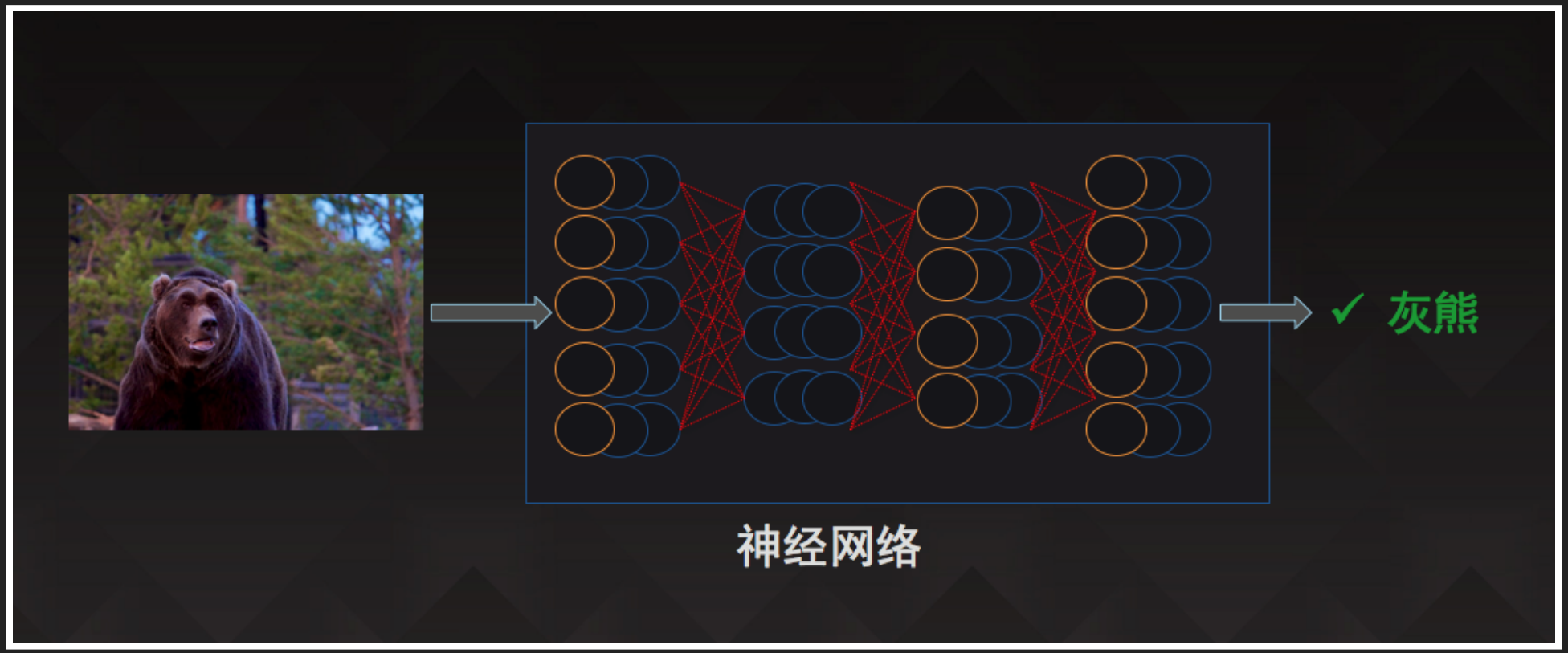
一组简单可以训练的数学单元集合，共同学习复杂的功能



深度学习 – 训练



深度学习 – 部署



深度学习 – 数据表现

- 表现层次
- 图片– 像素、主题、部分、轮廓、边缘等等
- 视频– 图像帧、每帧的像素、每一帧的deltas 值等等
- 文本– 字符、词、从句、句子等等
- 语音– 音频、频段、波长、调制等等

  …

深度学习的优势

- 特性自动推导和预期结果的优化调整
- 可变的自动学习的健壮性
- 重用性 – 相同的神经网络的方法可用于许多应用和数据类型
- 通过利用GPU的大规模并行计算 – 可扩展的大容量数据

深度学习的开发框架

- Torch (NYU,2002), Facebook AI, Google Deepmind
- Theano (University of Montreal, ~2010), 学院派
- Kersa, “Deep Learning library for Theano and TensorFlow”
- Caffe (Berkeley)，卷积神经网络,贾扬清
- TensorFlow (Google)
- Spark MLLib

深度学习中的开发框架框架

python has a wide range of deep learning-related libraries available

and of course: scikit learn

High level

Lasagne (theano-extension, models in python code, theano not hidden)

Keras (theano-wrapper, models in python code, abstracts theano away)

Pylearn2 (wrapper for theano, yaml, experiment-oriented)

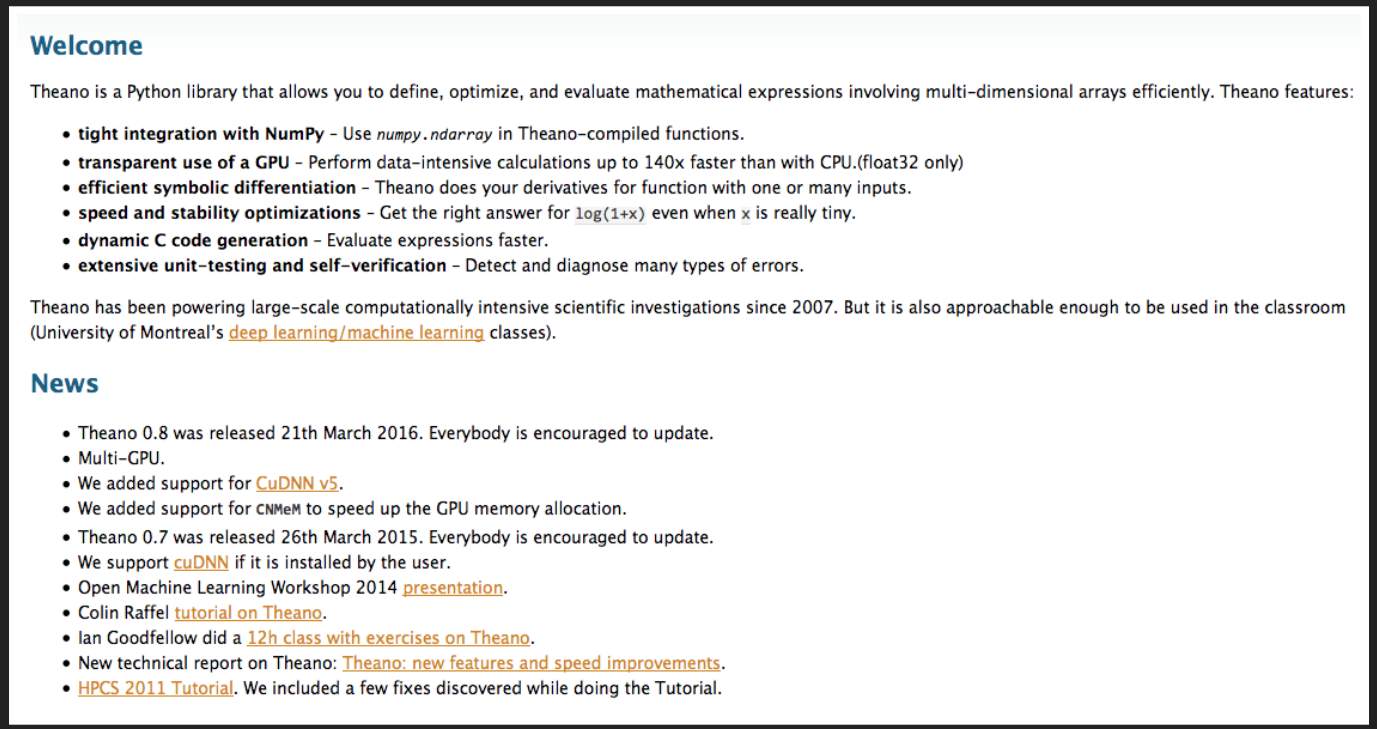
Caffe (computer-vision oriented DL framework, model-zoo, prototxt model definitions) pythonification ongoing!

Low level

theano (efficient gpu-powered math)

THEANO

- 学院派血统,Montreal University
- 非常灵活，非常复杂
- 通过底层借口可以做到大量的定制
- 衍生了大量的丰富的项目Keras, PyLearn2, Lasagne...
- Pythonic API, 非常好的文档



Welcome

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

- **tight integration with NumPy** – Use *numpy.ndarray* in Theano-compiled functions.
- **transparent use of a GPU** – Perform data-intensive calculations up to 140x faster than with CPU.(float32 only)
- **efficient symbolic differentiation** – Theano does your derivatives for function with one or many inputs.
- **speed and stability optimizations** – Get the right answer for `log(1+x)` even when `x` is really tiny.
- **dynamic C code generation** – Evaluate expressions faster.
- **extensive unit-testing and self-verification** – Detect and diagnose many types of errors.

Theano has been powering large-scale computationally intensive scientific investigations since 2007. But it is also approachable enough to be used in the classroom (University of Montreal's deep learning/machine learning classes).

News

- Theano 0.8 was released 21th March 2016. Everybody is encouraged to update.
- Multi-GPU.
- We added support for CuDNN v5.
- We added support for `CNMeM` to speed up the GPU memory allocation.
- Theano 0.7 was released 26th March 2015. Everybody is encouraged to update.
- We support cuDNN if it is installed by the user.
- Open Machine Learning Workshop 2014 presentation.
- Colin Raffel tutorial on Theano.
- Ian Goodfellow did a 12h class with exercises on Theano.
- New technical report on Theano: Theano: new features and speed improvements.
- HPCS 2011 Tutorial. We included a few fixes discovered while doing the Tutorial.

THEANO 实践

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

imports

theano symbolic variable initialization

our model

compiling to a python function

usage

THEANO 实践

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

metric to be optimized by model
learning signal for parameter(s)
how to change parameter based on learning signal

compiling to a python function

iterate through data 100 times and train model
on each example of input, output pairs

THEANO 中的卷积极神经网络

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

metric to be optimized by model
learning signal for parameter(s)
how to change parameter based on learning signal

compiling to a python function

iterate through data 100 times and train model
on each example of input, output pairs

为什么是 PYTHON？

最好的"胶水"代码用于研究、快速开发

iPython, 数据可视化

丰富的框架资源Theano, Kersa, TensorFlow

海量的社区、开源的支持

为什么需要 GPU?

Hardware [edit]

An early version of AlphaGo was tested on hardware with various numbers of CPUs and GPUs, running in asynchronous or distributed mode. ratings are listed below.[6] In the matches with more time per move higher ratings are achieved.

Configuration and performance

Configuration	Search threads	No. of CPU	No. of GPU	Elo rating
Single[6] p.10-11	40	48	1	2,151
Single	40	48	2	2,738
Single	40	48	4	2,850
Single	40	48	8	2,890
Distributed	12	428	64	2,937
Distributed	24	764	112	3,079
Distributed	40	1,202	176	3,140
Distributed	64	1,920	280	3,168

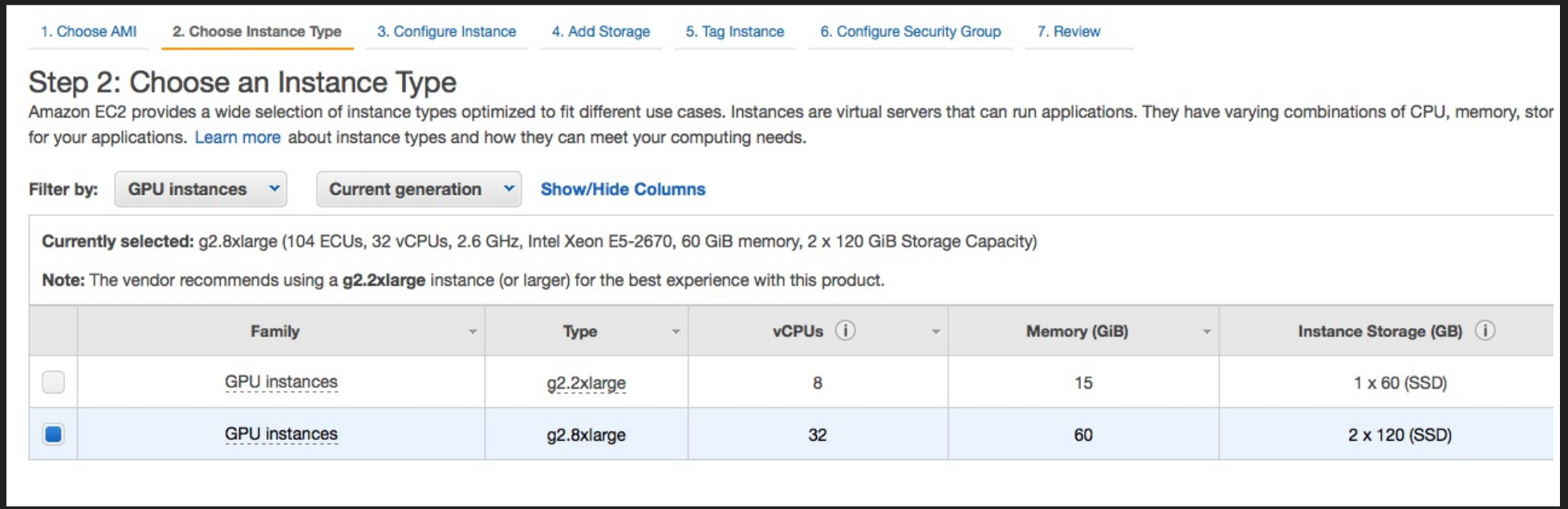
为什么需要 GPU?

- CPU - 指令并行执行，数据并行运算
- GPU - 矩阵类型的数值计算，尤其浮点运算

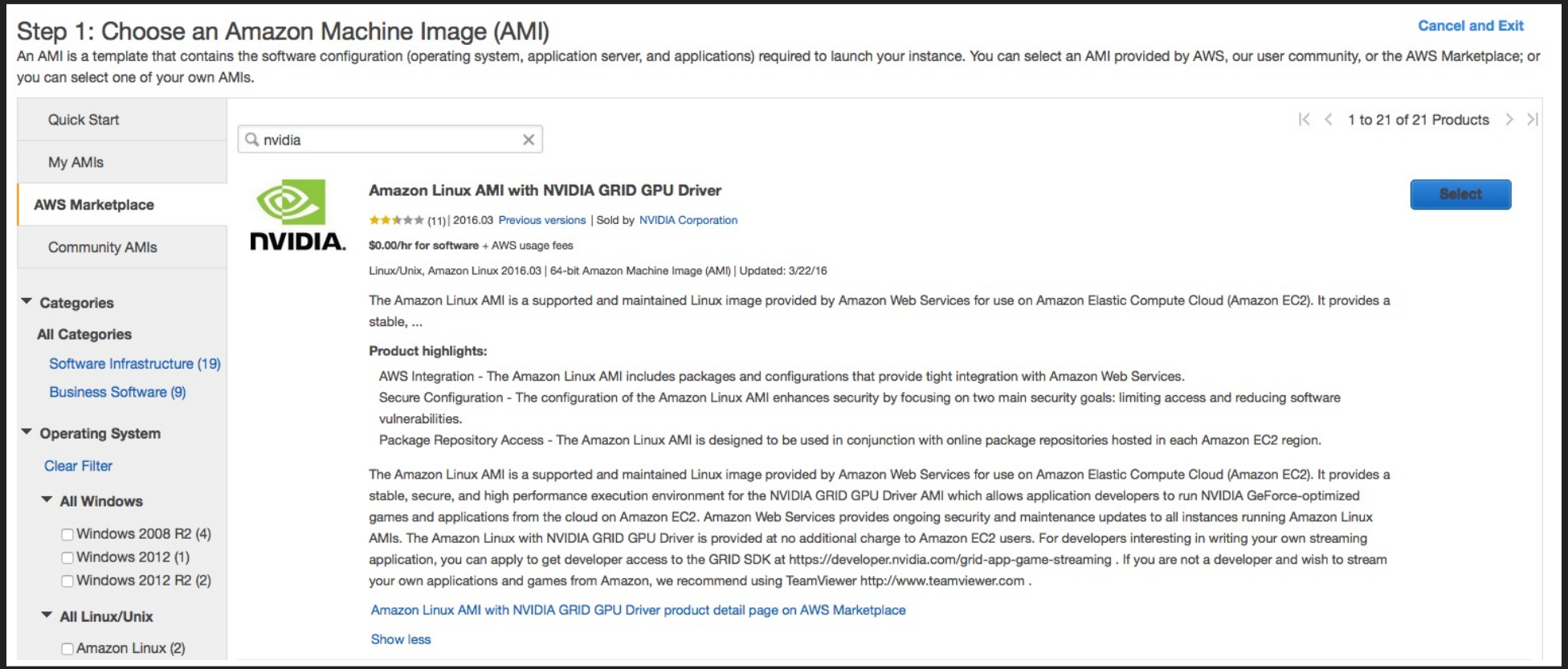
建立自己的深度学习的应用环境

适用于 深度学习的 AWS G2实例 –

- 4个NVIDIA GRID GPUs, 每1个包括了 1,536 CUDA cores 以及 4 GB of video
- 32 vCPUs
- 60 GB内存
- 240 GB (2 x 120) of SSD 存储



深度学习的应用环境


Step 1: Choose an Amazon Machine Image (AMI)
Cancel and Exit
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.
Quick Start
My AMIs
AWS Marketplace
Community AMIs
Categories
All Categories
Software Infrastructure (19)
Business Software (9)
Operating System
Clear Filter
All Windows
Windows 2008 R2 (4)
Windows 2012 (1)
Windows 2012 R2 (2)
All Linux/Unix
Amazon Linux (2)
nvidia
1 to 21 of 21 Products
Amazon Linux AMI with NVIDIA GRID GPU Driver
Select
(11) | 2016.03 Previous versions | Sold by NVIDIA Corporation
$0.00/hr for software + AWS usage fees
Linux/Unix, Amazon Linux 2016.03 | 64-bit Amazon Machine Image (AMI) | Updated: 3/22/16
The Amazon Linux AMI is a supported and maintained Linux image provided by Amazon Web Services for use on Amazon Elastic Compute Cloud (Amazon EC2). It provides a stable, ...
Product highlights:
AWS Integration - The Amazon Linux AMI includes packages and configurations that provide tight integration with Amazon Web Services.
Secure Configuration - The configuration of the Amazon Linux AMI enhances security by focusing on two main security goals: limiting access and reducing software vulnerabilities.
Package Repository Access - The Amazon Linux AMI is designed to be used in conjunction with online package repositories hosted in each Amazon EC2 region.
The Amazon Linux AMI is a supported and maintained Linux image provided by Amazon Web Services for use on Amazon Elastic Compute Cloud (Amazon EC2). It provides a stable, secure, and high performance execution environment for the NVIDIA GRID GPU Driver AMI which allows application developers to run NVIDIA GeForce-optimized games and applications from the cloud on Amazon EC2. Amazon Web Services provides ongoing security and maintenance updates to all instances running Amazon Linux AMIs. The Amazon Linux with NVIDIA GRID GPU Driver is provided at no additional charge to Amazon EC2 users. For developers interesting in writing your own streaming application, you can apply to get developer access to the GRID SDK at https://developer.nvidia.com/grid-app-game-streaming . If you are not a developer and wish to stream your own applications and games from Amazon, we recommend using TeamViewer http://www.teamviewer.com .
Amazon Linux AMI with NVIDIA GRID GPU Driver product detail page on AWS Marketplace
Show less

深度学习的应用环境 - 安装和配置

```
sudo yum updat

sudo yum install git python-nose gcc gcc-gfortran
gcc-c++ blas-devel lapack-devel atlas-devel
python34-devel python3.4  lapack64-devel python34-pip
python34-virtualenv

pip-3.4 install Theano numpy scipy nose keras pycuda
```

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions

export PATH=/opt/nvidia/cuda/bin:$PATH
export LD_LIBRARY_PATH=/opt/nvidia/cuda/lib64
```

深度学习的应用环境 – 检查GPU

```
[ec2-user@ip-10-0-0-185 ~]$ nvidia-smi
Wed Apr 20 02:57:18 2016
+------------------------------------------------------+
| NVIDIA-SMI 340.32     Driver Version: 340.32         |
|-------------------------------+----------------------+----------------------+
| GPU  Name         Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GRID K520           On   | 0000:00:03.0     Off |                  N/A |
| N/A   33C    P8    17W / 125W |     10MiB /  4095MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GRID K520           On   | 0000:00:04.0     Off |                  N/A |
| N/A   28C    P8    17W / 125W |     10MiB /  4095MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  GRID K520           On   | 0000:00:05.0     Off |                  N/A |
| N/A   31C    P8    18W / 125W |     10MiB /  4095MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  GRID K520           On   | 0000:00:06.0     Off |                  N/A |
| N/A   27C    P8    17W / 125W |     10MiB /  4095MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

深度学习的应用环境 – THEANO

~/.theanorc

```
[global]
floatX=float32
device=gpu
[mode]=FAST_RUN

[nvcc]
fastmath=True

[cuda]
root=/opt/nvidia/cuda
```

深度学习的应用环境 – THEANO

我的第一个Theano 程序

```
 1 from theano import function, config, shared, sandbox
 2 import theano.tensor as T
 3 import numpy
 4 import time
 5
 6 vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
 7 iters = 1000
 8
 9 rng = numpy.random.RandomState(22)
10 x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
11 f = function([], T.exp(x))
12 print(f.maker.fgraph.toposort())
13 t0 = time.time()
14 for i in range(iters):
15     r = f()
16 t1 = time.time()
17 print("Looping %d times took %f seconds" % (iters, t1 - t0))
18 print("Result is %s" % (r,))
19 if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
20     print('Used the cpu')
21 else:
22     print('Used the gpu')
23
```

深度学习的应用环境 – THEANO

我的第一个Theano 程序

```
from theano import function, config, shared, sandbox
import theano.tensor as T
import numpy
import time

vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
iters = 1000

rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], T.exp(x))
print(f.maker.fgraph.toposort())
t0 = time.time()
for i in range(iters):
    r = f()
t1 = time.time()
print("Looping %d times took %f seconds" % (iters, t1 - t0))
print("Result is %s" % (r,))
if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
    print('Used the cpu')
else:
    print('Used the gpu')
```

深度学习的应用环境 – THEANO

GPU vs. CPU

```
(Theano)[ec2-user@ip-10-0-0-185 ~]$ THEANO_FLAGS=mode=FAST_RUN,device=gpu,floatX
=float32 python3.4 gpu_test.py
Using gpu device 0: GRID K520 (CNMeM is disabled, CuDNN not available)
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>), HostFromGpu(Gp
uElemwise{exp,no_inplace}.0)]
Looping 1000 times took 0.784566 seconds
Result is [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
  1.62323296]
Used the gpu
(Theano)[ec2-user@ip-10-0-0-185 ~]$ THEANO_FLAGS=mode=FAST_RUN,device=cpu,floatX
=float32 python3.4 gpu_test.py
[Elemwise{exp,no_inplace}(<TensorType(float32, vector)>)]
Looping 1000 times took 2.953590 seconds
Result is [ 1.23178029  1.61879337  1.52278066 ...,  2.20771813  2.29967761
  1.62323284]
Used the cpu
```

TENSORFLOW 的新进展

分布式的深度学习框架

Announcing TensorFlow 0.8 – now with distributed computing support!

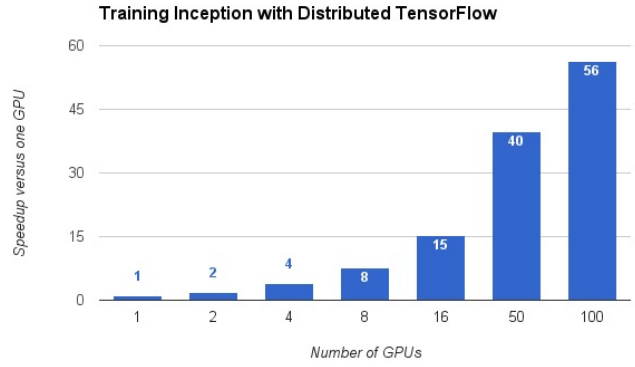
Wednesday, April 13, 2016

Posted by Derek Murray, Software Engineer

Google uses machine learning across a wide range of its products. In order to continually improve our models, it's crucial that the training process be as fast as possible. One way to do this is to run TensorFlow across hundreds of machines, which shortens the training process for some models from weeks to hours, and allows us to experiment with models of increasing size and sophistication. Ever since we released TensorFlow as an open-source project, distributed training support has been one of the most requested features. Now the wait is over.

Today, we're excited to release TensorFlow 0.8 with distributed computing support, including everything you need to train distributed models on your own infrastructure. Distributed TensorFlow is powered by the high-performance gRPC library, which supports training on hundreds of machines in parallel. It complements our recent announcement of Google Cloud Machine Learning, which enables you to train and serve your TensorFlow models using the power of the Google Cloud Platform.

To coincide with the TensorFlow 0.8 release, we have published a distributed trainer for the Inception image classification neural network in the TensorFlow models repository. Using the distributed trainer, we trained the Inception network to 78% accuracy in less than 65 hours using 100 GPUs. Even small clusters—or a couple of machines under your desk—can benefit from distributed TensorFlow, since adding more GPUs improves the overall throughput, and produces accurate results sooner.



TENSORFLOW 的新进展

分布式的深度学习框架

Announcing TensorFlow 0.8 – now with distributed computing support!

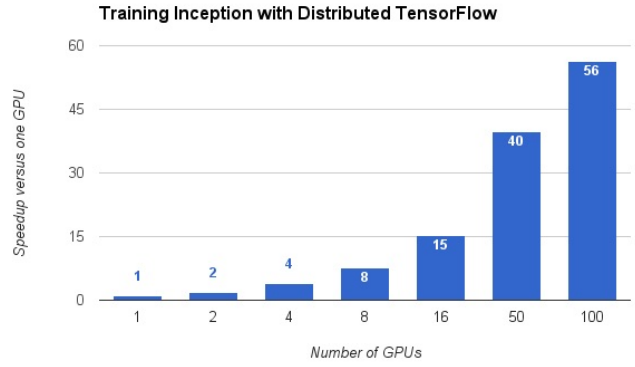
Wednesday, April 13, 2016

Posted by Derek Murray, Software Engineer

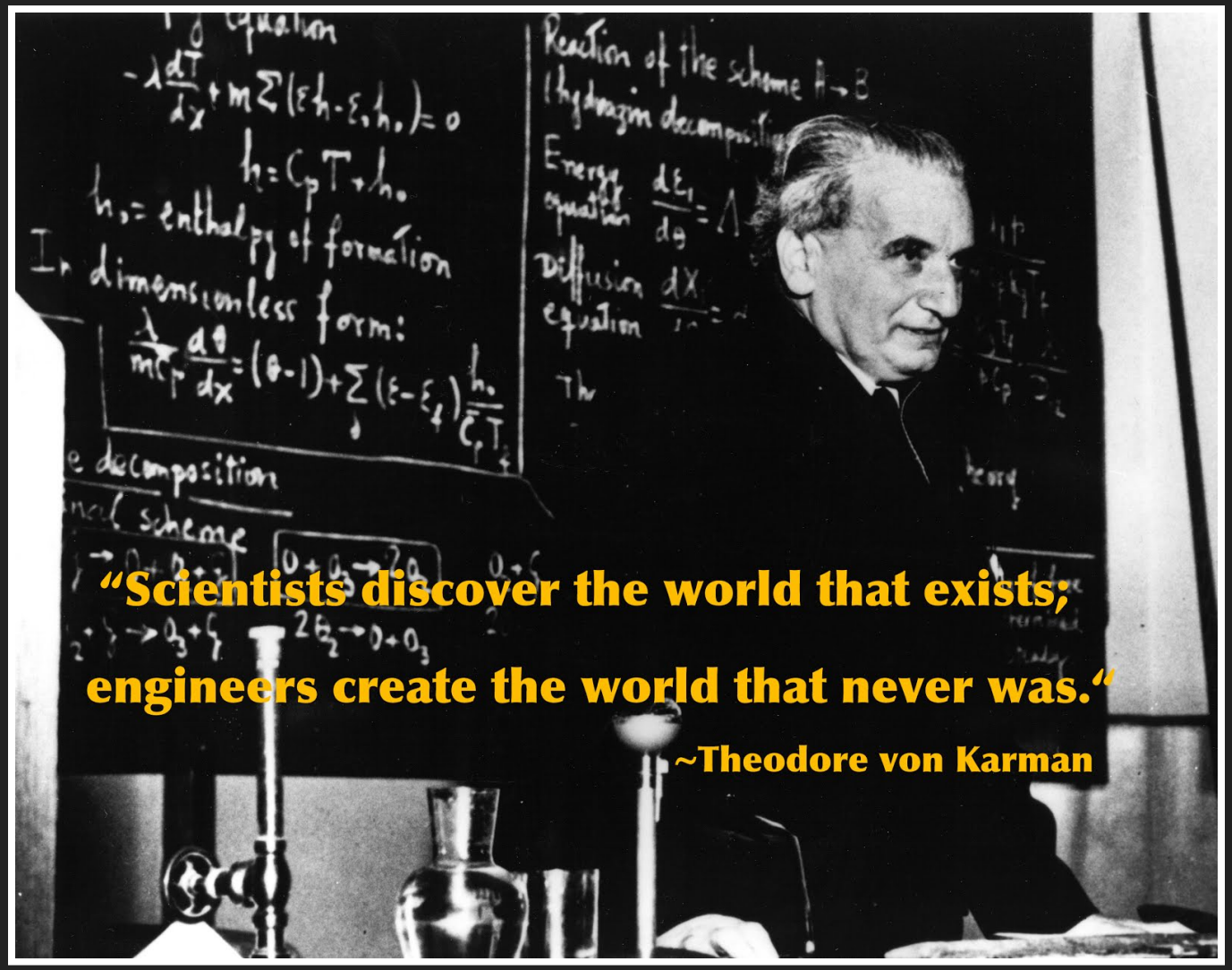
Google uses machine learning across a wide range of its products. In order to continually improve our models, it's crucial that the training process be as fast as possible. One way to do this is to run TensorFlow across hundreds of machines, which shortens the training process for some models from weeks to hours, and allows us to experiment with models of increasing size and sophistication. Ever since we released TensorFlow as an open-source project, distributed training support has been one of the most requested features. Now the wait is over.

Today, we're excited to release TensorFlow 0.8 with distributed computing support, including everything you need to train distributed models on your own infrastructure. Distributed TensorFlow is powered by the high-performance gRPC library, which supports training on hundreds of machines in parallel. It complements our recent announcement of Google Cloud Machine Learning, which enables you to train and serve your TensorFlow models using the power of the Google Cloud Platform.

To coincide with the TensorFlow 0.8 release, we have published a distributed trainer for the Inception image classification neural network in the TensorFlow models repository. Using the distributed trainer, we trained the Inception network to 78% accuracy in less than 65 hours using 100 GPUs. Even small clusters—or a couple of machines under your desk—can benefit from distributed TensorFlow, since adding more GPUs improves the overall throughput, and produces accurate results sooner.



工程化思维 vs. 科学化思维



THINK GREAT THOUGHTS AND YOU WILL BE GREAT.

心怀伟大，你将会变得伟大!

谢谢！