A brief introduction to PyTorch

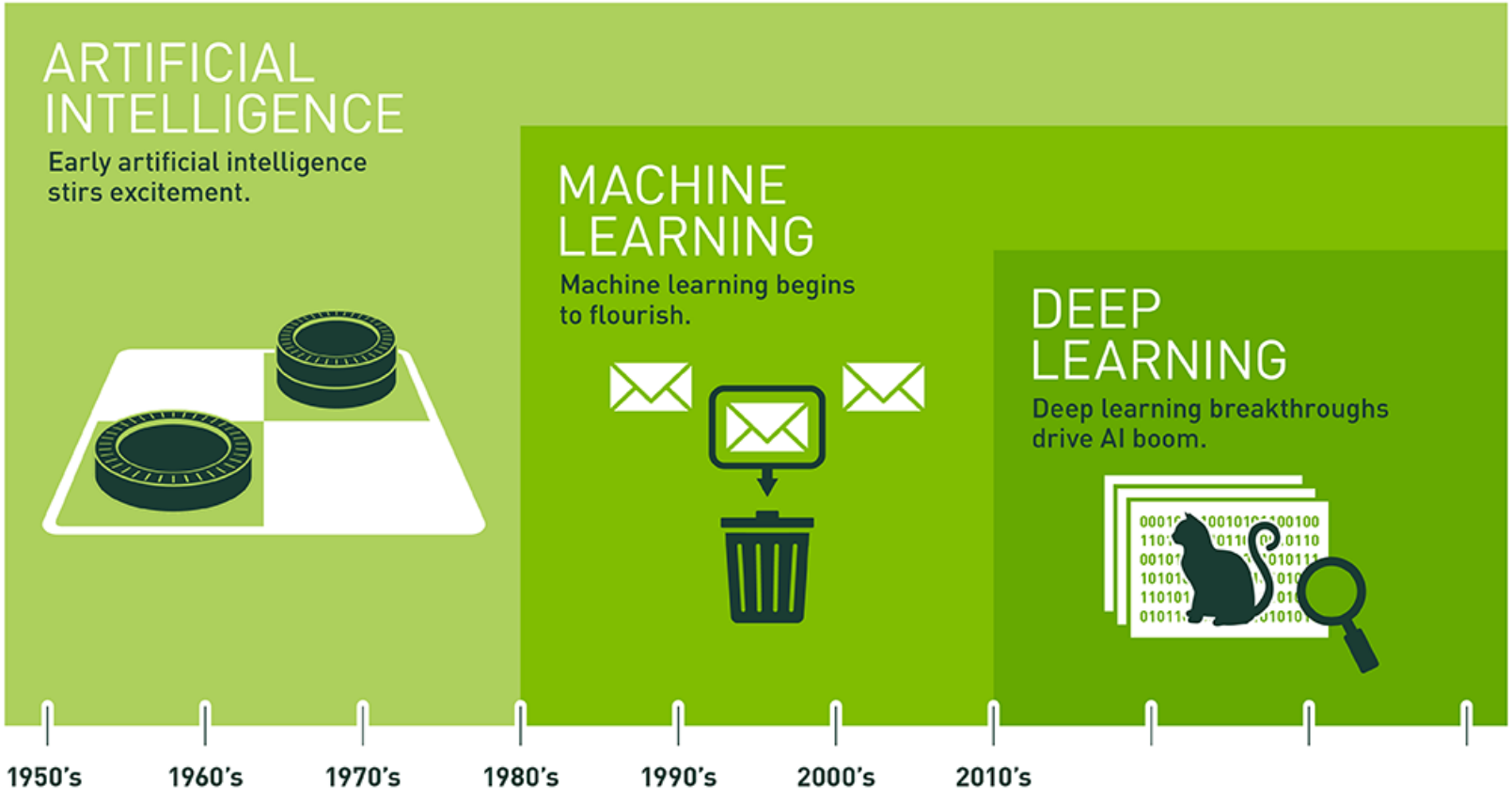
@lpyug 2018-02-13 19:00.

Martin Czygan | pytorch-tour

What is Deep Learning?

> Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations.

AI, ML, DL, ...



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.

From: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

What is Deep Learning?

- some definition: anything with more than two hidden layers
- computationally expensive, high capacity learning machines

Advances

- AlphaGo (March 2016, Deep Learning and the Game of Go)
- ImageNet classification (2014, VGG16, VGG19)
- Real-Time object detection (2013, darknet; 2016, YOLO)
- Image Captioning
- Neural Style Transfer
- WaveNet (speech generation)
- Speech recognition (2017, DeepSpeech)
- Translation (2016, OpenNMT)
- Word Embeddings (2013, word2vec; 2016, fasttext)

And much more

- Image and scene generation
- Image segmentation
- Lip reading
- Text generation
- Time series forecast

A model zoo

- http://www.asimovinstitute.org/wp-content/uploads/2016/09/neuralnetworks.png

History: "A tiny bit of money"

In Nov 2007, Geoffrey Hinton gave a tech talk at Google, called *The Next Generation of Neural Networks*. He seems like a slightly desperate.

> We only trained this network once one one data set. If we could get a tiny bit of money from someone we could make this whole thing work much better.

Ten years later Hinton introduces the Vector Institute at University of Toronto.

Why Now?

- In short: data, cuda, relu.
- Or: availability of data, GPUS, algorithmic advances.

Deep Learning Frameworks

- Abstract away the neural network construction and learning algorithms
- Lots of Python wrappers or pure Python APIs
- tensorflow, keras, mxnet, pytorch, paddle, CNTK, dlib, Theano, chainer, dynet, ...
- Other languages: caffe, caffe2, DL4J, DIGITS

Deep Learning Frameworks

- gpu*
- computational graph*
- automatic differentiation*
- optimization algorithms
- utilities: data loading, dashboard, ...

Parts and Ingredients

Build a computational graph, utilize automatic differentiation, to adjust the parameters of your model according to a given loss function, that captures the distance between the computed and the expected output, given enough training data.

Overview

- basics
- differentiation
- NN
- examples

NumPy performance

- Notebook #1

PyTorch Basics

It's a Python based scientific computing package targeted at two sets of audiences:

- A replacement for NumPy to use the power of GPUs
- a deep learning research platform that provides maximum flexibility and speed

Basic data structures

- Tensors: like np.ndarray but, GPU-enabled
- Notebook #2

Automatic Differentiation

> Automatic differentiation (AD) is software to transform code for one function into code for the derivative of the function.

Gradients

> Imagine you want to test out a new machine learning model for your data. This usually means coming up with some loss function to capture how well your model fits the data and optimizing that loss with respect to the model parameters.

> If there are many model parameters (neural nets can have millions) then you need gradients. You then have two options:
>
> - derive and code them up yourself, or
> - implement your model using the syntactic and semantic constraints of a system like Theano or TensorFlow.

Automatic Differentiation

Three ways to find the derivative:

- numerical approximation
- symbolic calculation
- AD

Numeric

Table 6.5 Numerical Approximations to $f''(x)$ for Example 6.4

Step size	Approximation by formula (6)	Error using formula (6)
$h = 0.1$	−0.696126300	−0.000580409
$h = 0.01$	−0.696690000	−0.000016709
$h = 0.001$	−0.696000000	−0.000706709

Symbolic

An example of the sort of rules we will wish to specify is the following set for algebraic simplification.

```
(define algebra-rules
 '(
   ( ((? op) (?c e1) (?c e2))          (: (op e1 e2))                )
   ( ((? op) (? e1) (?c e2))           ((: op) (: e2) (: e1))        )
   ( (+ 0 (? e))                       (: e)                         )
   ( (* 1 (? e))                       (: e)                         )
   ( (* 0 (? e))                       0                             )
   ( (* (?c e1) (* (?c e2) (? e3)))    (* (: (* e1 e2)) (: e3))      )
   ( (* (? e1) (* (?c e2) (? e3)))     (* (: e2) (* (: e1) (: e3)))  )
   ( (* (* (? e1) (? e2)) (? e3))      (* (: e1) (* (: e2) (: e3)))  )
   ( (+ (?c e1) (+ (?c e2) (? e3)))    (+ (: (+ e1 e2)) (: e3))      )
   ( (+ (? e1) (+ (?c e2) (? e3)))     (+ (: e2) (+ (: e1) (: e3)))  )
   ( (+ (+ (? e1) (? e2)) (? e3))      (+ (: e1) (+ (: e2) (: e3)))  )
   ( (+ (* (?c c) (? a)) (* (?c d) (? a)))
                                       (* (: (+ c d)) (: a))         )
   ( (* (? c) (+ (? d) (? e)))         (+ (* (: c) (: d))
                                          (* (: c) (: e)))           )
    ))
```

Automatic

- also called algorithmic differentiation
- there are standalone libraries for this, e.g. autograd

Example with autograd: Notebook #3

Automatic

> To compute the gradient, Autograd first has to record every transformation that was applied to the input as it was turned into the output of your function. To do this, Autograd wraps functions (using the function primitive) so that when they're called, they add themselves to a list of operations performed. Autograd's core has a table mapping these wrapped primitives to their corresponding gradient functions.

Automatic

```
87  # ----- Simple grads -----
88  defjvp(anp.abs,
89      lambda g, ans, x : anp.real(g * replace_zero(anp.conj(x), 0.)) / replace_zero(ans, 1.))
90  defjvp(anp.fabs,        lambda g, ans, x : anp.sign(x) * g)  # fabs doesn't take complex numbers.
91  defjvp(anp.absolute,    lambda g, ans, x : anp.real(g * anp.conj(x)) / ans)
92  defjvp(anp.reciprocal,  lambda g, ans, x : - g / x**2)
93  defjvp(anp.exp,         lambda g, ans, x : ans * g)
94  defjvp(anp.exp2,        lambda g, ans, x : ans * anp.log(2) * g)
95  defjvp(anp.expm1,       lambda g, ans, x : (ans + 1) * g)
96  defjvp(anp.log,         lambda g, ans, x : g / x)
97  defjvp(anp.log2,        lambda g, ans, x : g / x / anp.log(2))
98  defjvp(anp.log10,       lambda g, ans, x : g / x / anp.log(10))
99  defjvp(anp.log1p,       lambda g, ans, x : g / (x + 1))
100 defjvp(anp.sin,         lambda g, ans, x : g * anp.cos(x))
101 defjvp(anp.cos,         lambda g, ans, x : - g * anp.sin(x))
102 defjvp(anp.tan,         lambda g, ans, x : g / anp.cos(x) **2)
```

Autograd in PyTorch

> The autograd package provides automatic differentiation for all operations on Tensors.

- Notebook #3

Simple MLP

- Notebook #4

CIFAR10 Example

- Notebook #5

...