

DEEP
LEARNING
INSTITUTE

第100回お試しアカウント付き並列プログラミング講習会
「REEDBUSH スパコンを用いたGPUディープラーニング入門」

ディープラーニング基礎

山崎和博

NVIDIA, ディープラーニング ソリューションアーキテクト

AGENDA

ディープラーニングとは

ディープラーニングの基本と処理の流れ

ディープラーニングを支えるソフトウェア

本日使用するフレームワーク:
Chainer&ChainerMN

ディープラーニングとは



ディープラーニングは機械学習の一分野

人工知能 (AI)



マシンラーニング
(機械学習)

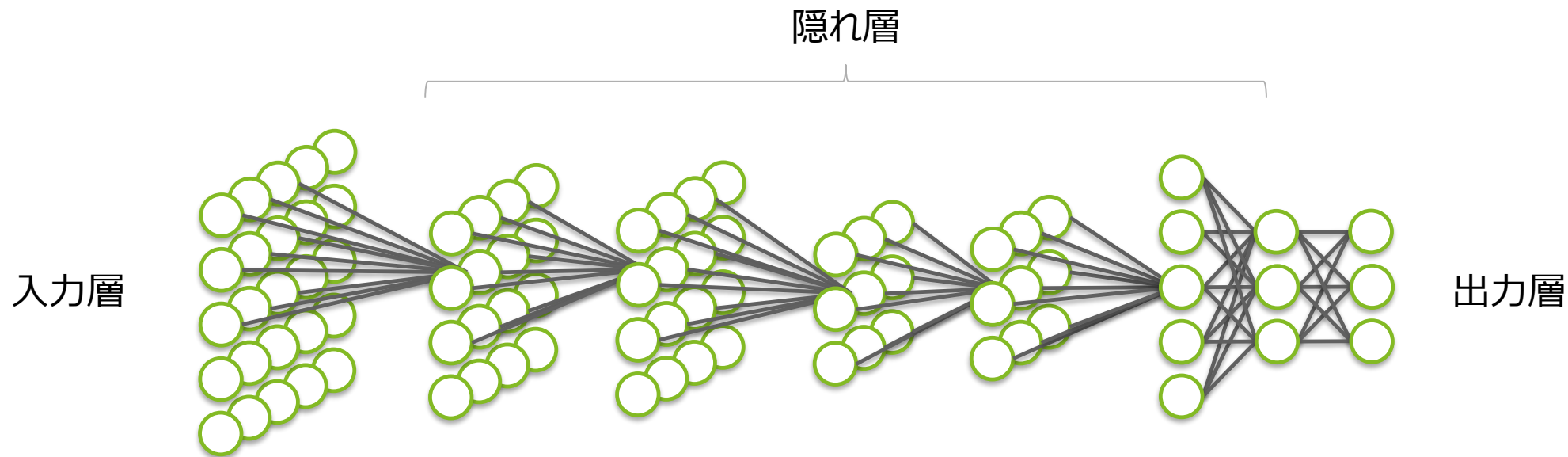


ディープラーニング
(深層学習)



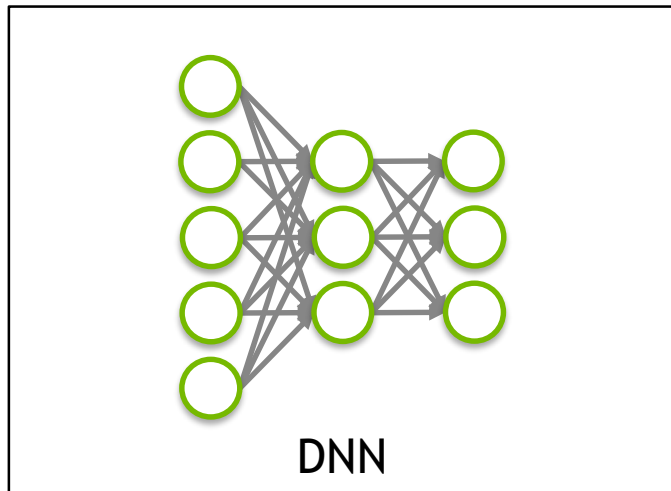
ディープラーニングとは

多数の層から構成されるニューラルネットワークがベース



十分なトレーニングデータを与え学習させることで
複雑な問題を解くことができるようになる

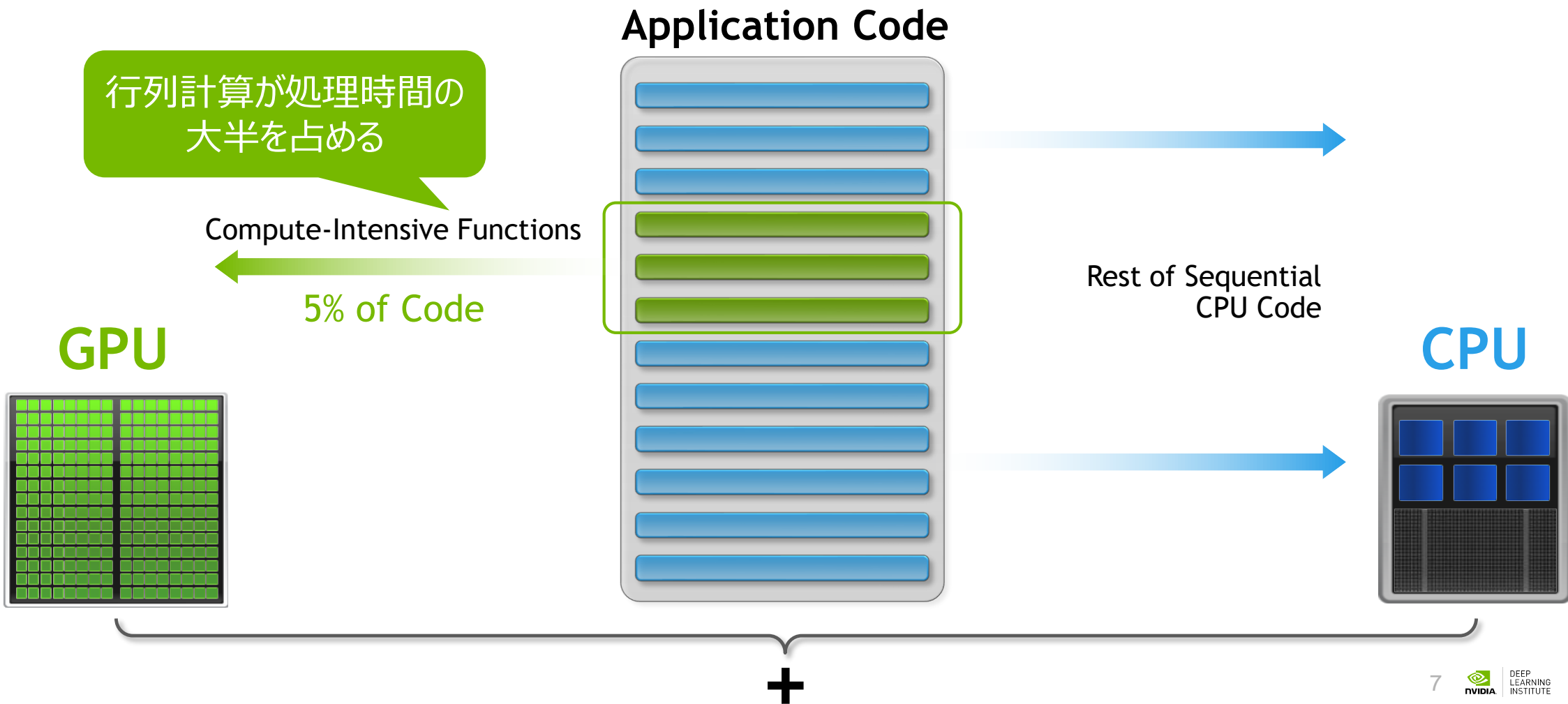
ディープラーニングを加速する3つの要因



“Google’s AI engine also reflects how the world of computer hardware is changing. (It) depends on machines equipped with GPUs... And it depends on these chips more than the larger tech universe realizes.”

WIRED

どのようにGPUはディープラーニングを加速させるか

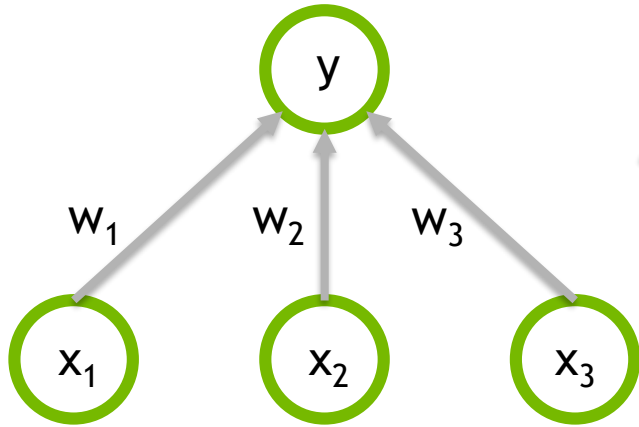


ディープラーニングの基本と処理の流れ

人工ニューラルネットワーク

基本単位

人工ニューロン (パーセプトロン)



$$y = w_1x_1 + w_2x_2 + w_3x_3$$

人工ニューロンを
たくさんの層として組み合わせる

各層にそれぞれ
重みパラメータを持つ

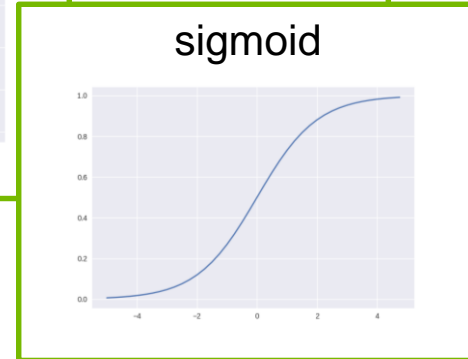
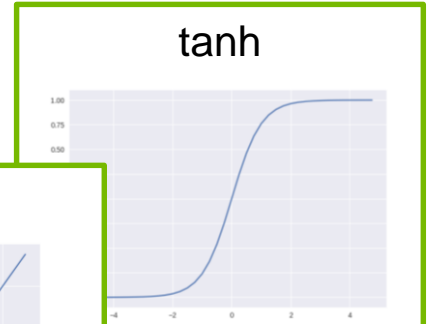
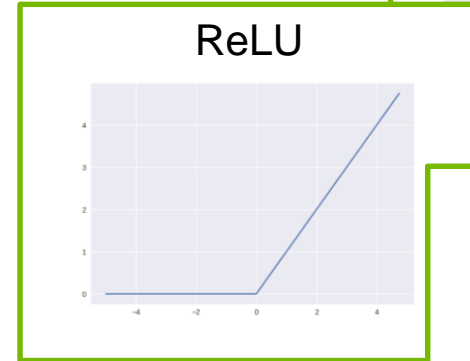
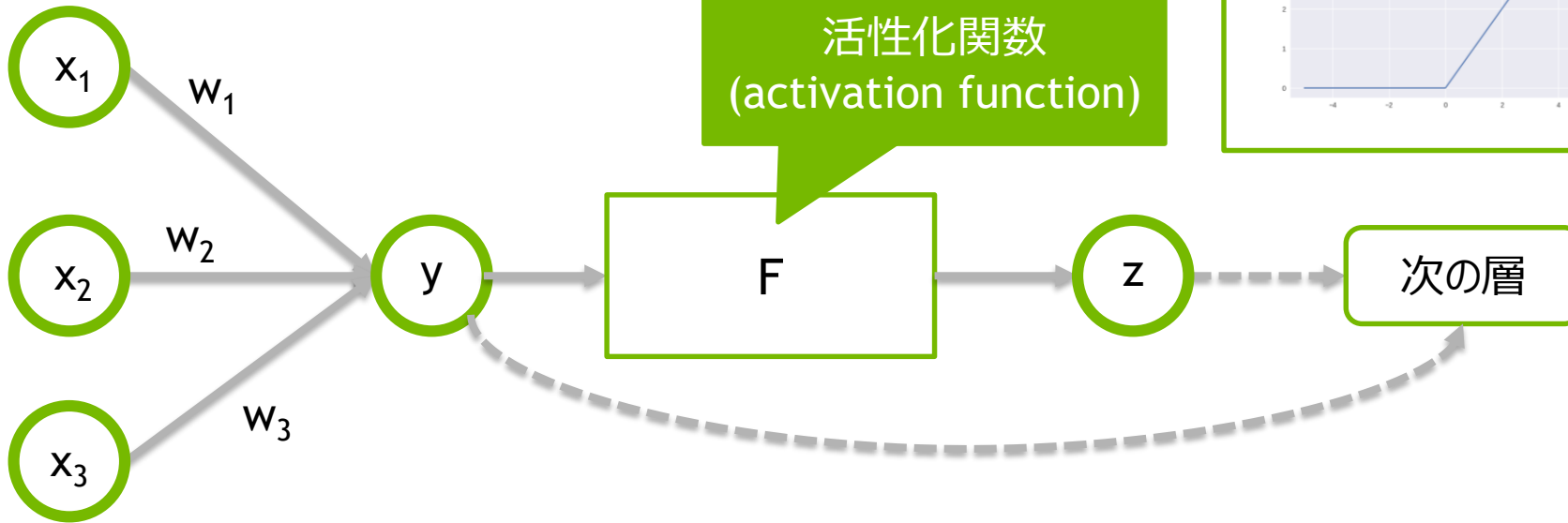


人工ニューラルネットワーク

人工ニューラルネットワーク

活性化関数

人工ニューロン (パーセプトロン)



$$y = w_1x_1 + w_2x_2 + w_3x_3$$



$$z = F(y)$$

F: activation function

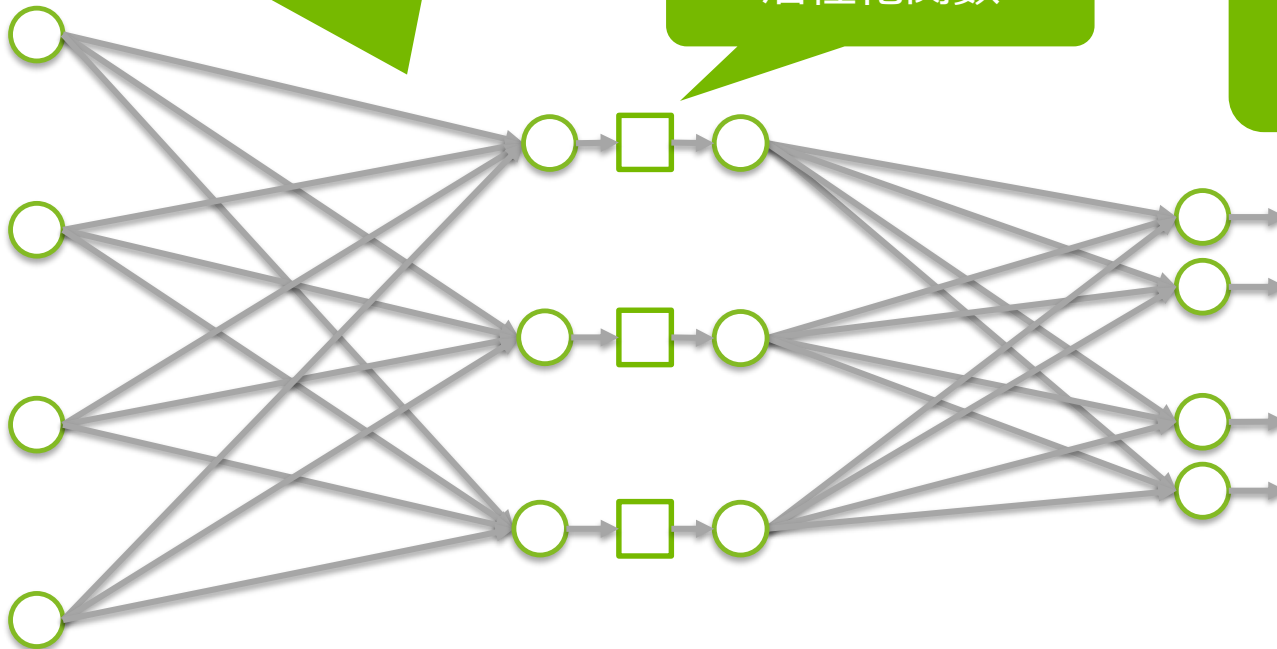
人工ニューラルネットワーク

基本構造のまとめ

各ユニット（ノードとも）の出力は、
各層の重みと積算され次の層へ

活性化関数

各ノードは
入力された値を合計して出力



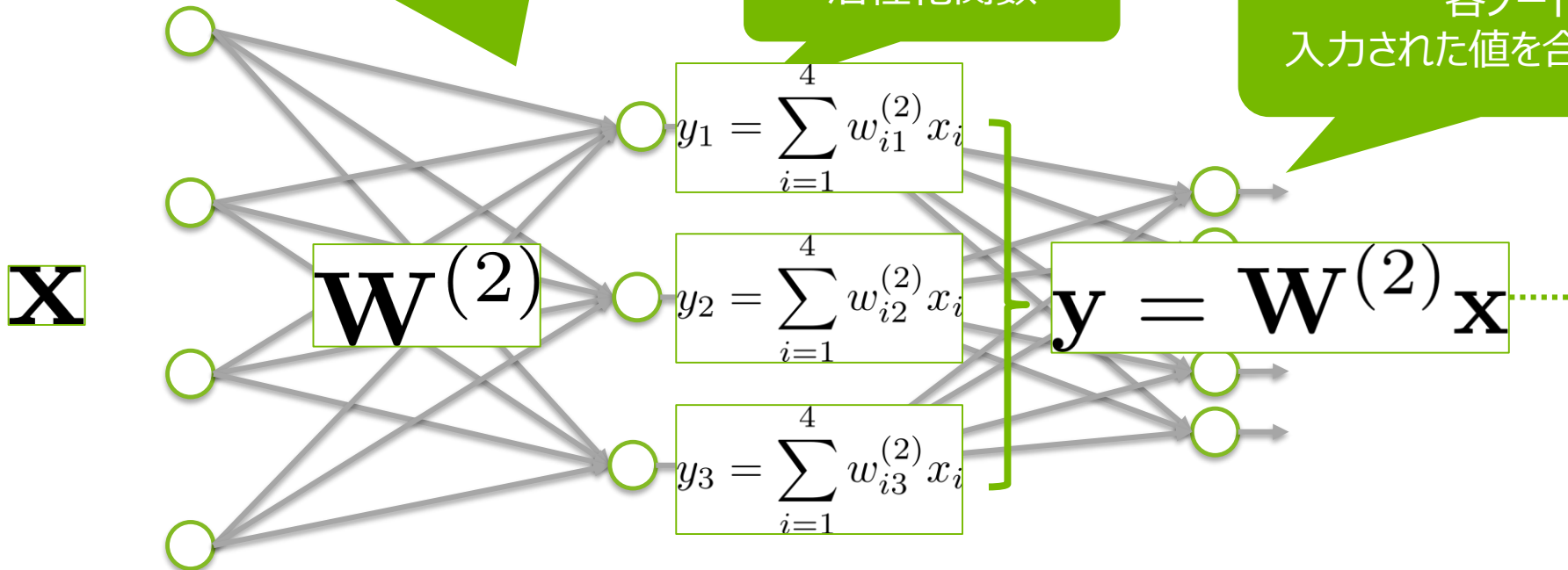
人工ニューラルネットワーク

基本構造のまとめ

各ユニット（ノードとも）の出力は、
各層の重みと積算され次の層へ

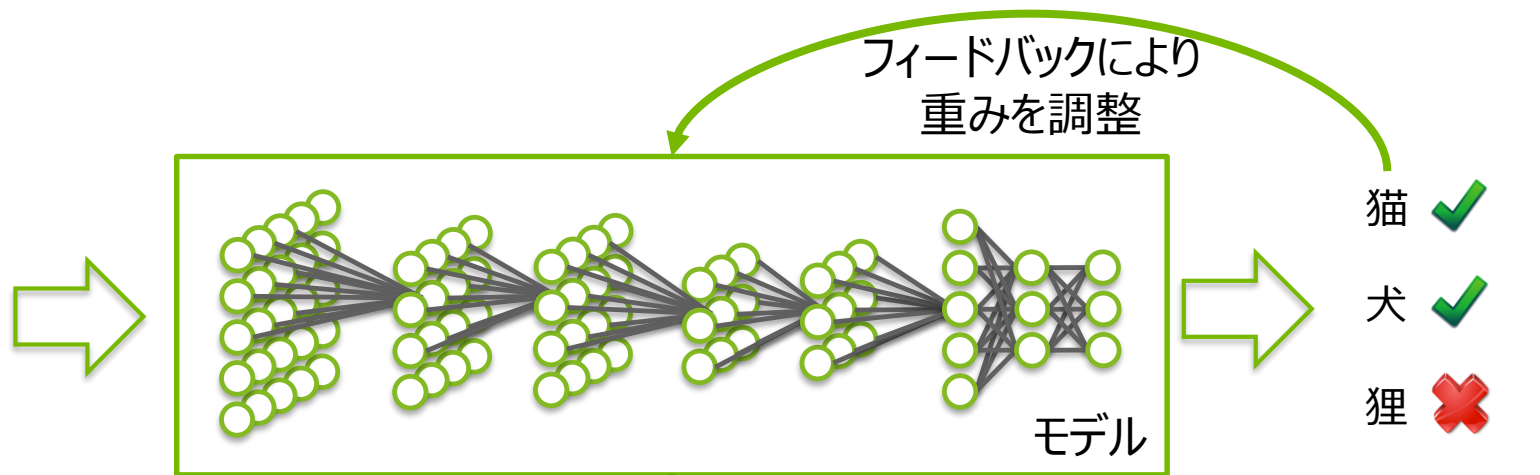
活性化関数

各ノードは
入力された値を合計して出力

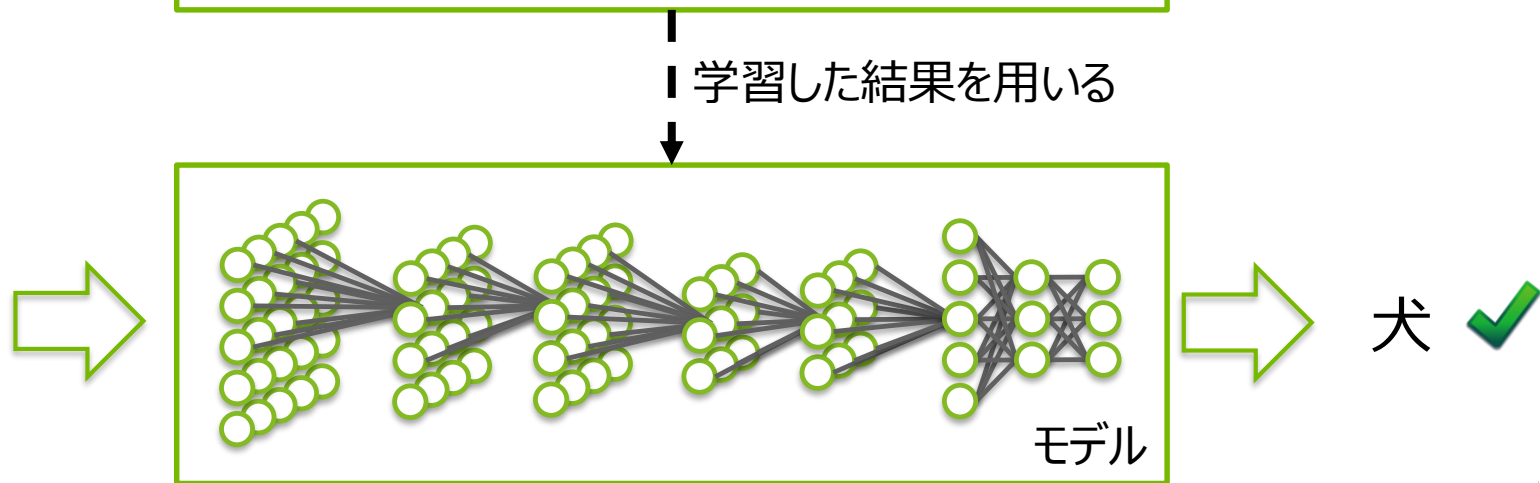


ディープラーニングの2つのフェーズ

学習(training):



推論(inference):



ディープラーニングの学習フロー

トレーニングデータ



4. 誤差が小さくなるまで繰り返す

1. 入力されたデータにしたがい結果を出力

出力

“cat”
“human”
“tiger”

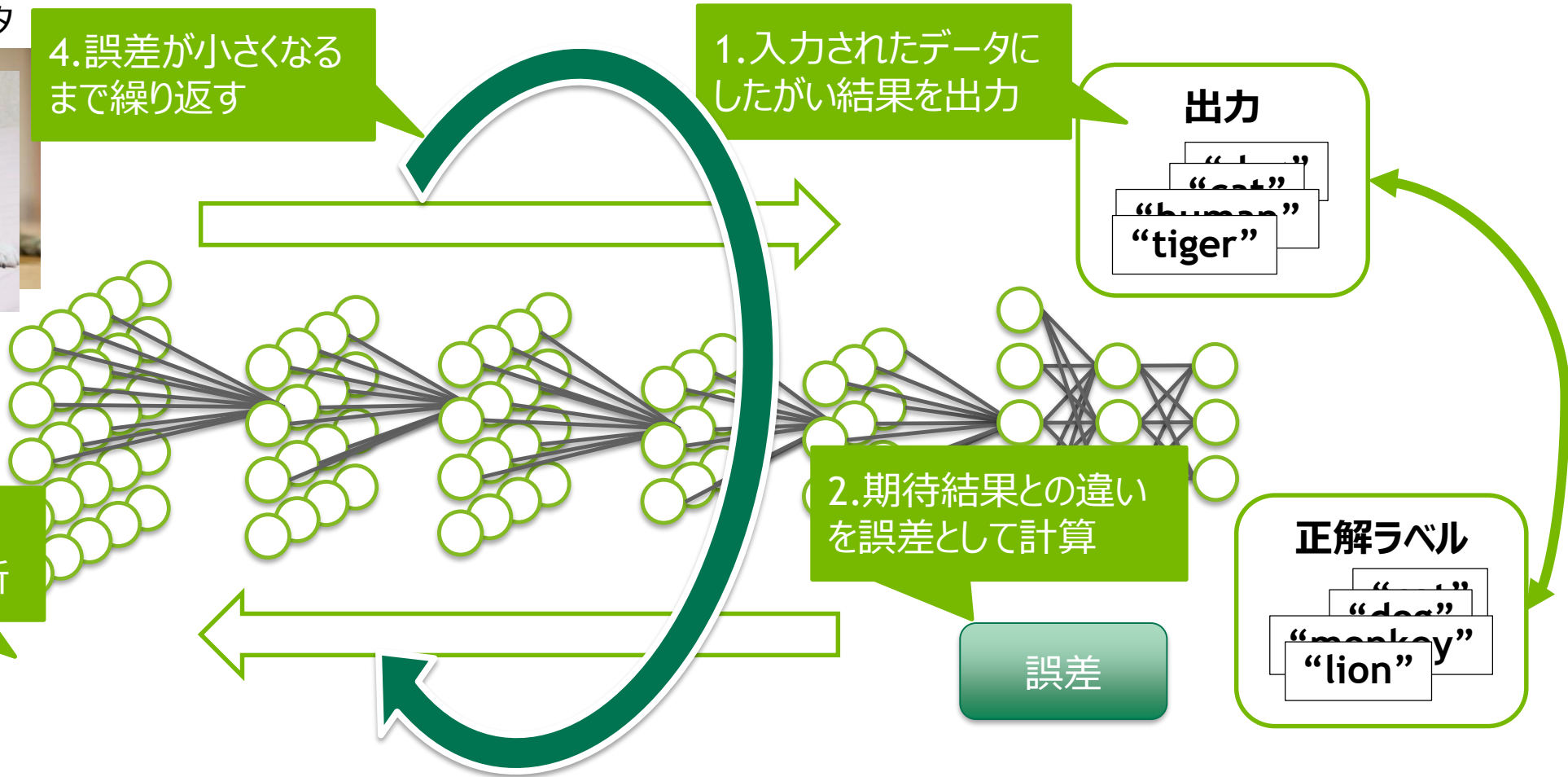
2. 期待結果との違いを誤差として計算

誤差

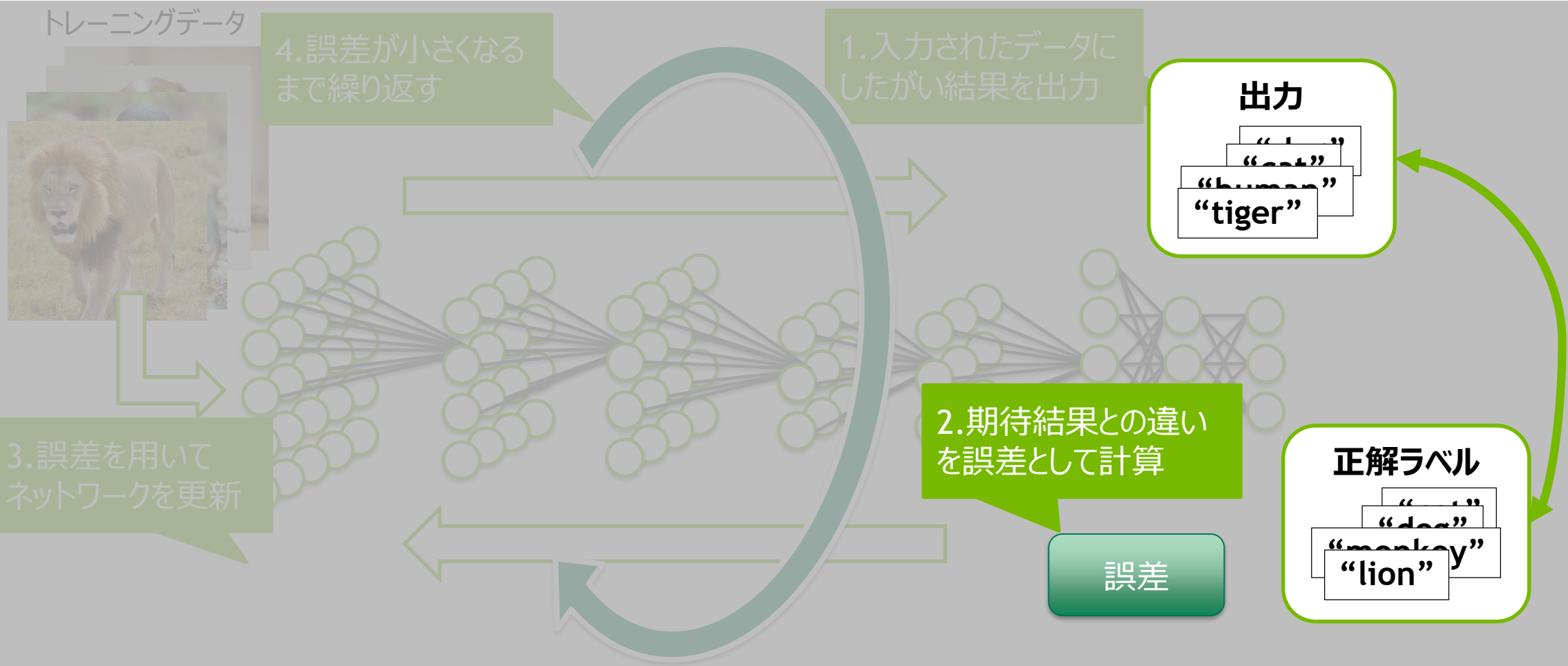
正解ラベル

“dog”
“monkey”
“lion”

3. 誤差を用いてネットワークを更新

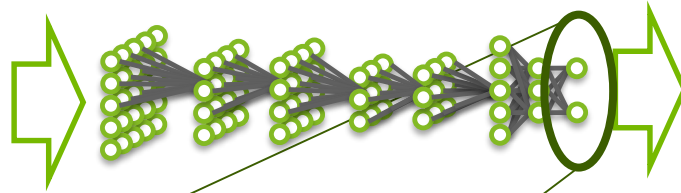


ディープラーニングの学習フロー



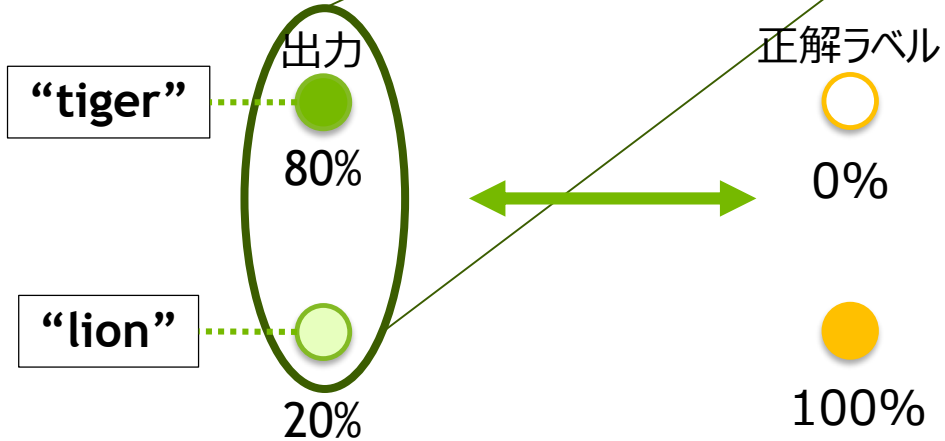
誤差の計算

トレーニングデータ



出力

“tiger” 80%
“lion” 20%

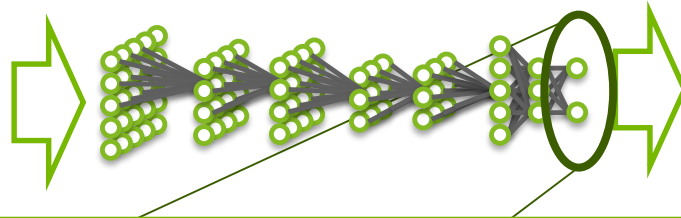


誤差

期待結果との差などを
誤差として計算

誤差の計算

トレーニングデータ



出力

“tiger” 80%

“lion” 20%

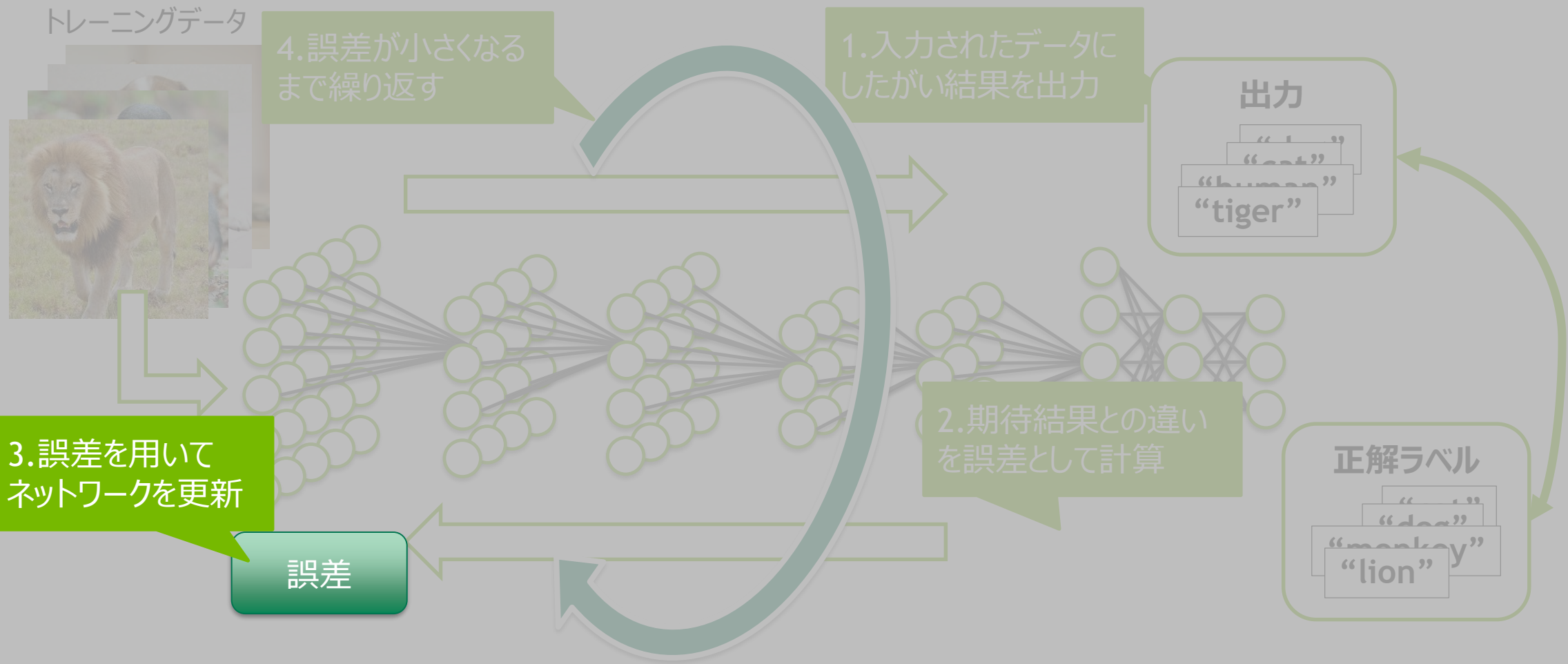
誤差の例: 交差エントロピー (cross entropy)

$$E = - \left(d_1 \log z_1(\mathbf{x}) + d_2 \log z_2(\mathbf{x}) \right)$$

d_1 : 入力 tiger であることを示す正解 (0 or 1) d_2 : 入力 lion であることを示す正解 (0 or 1)

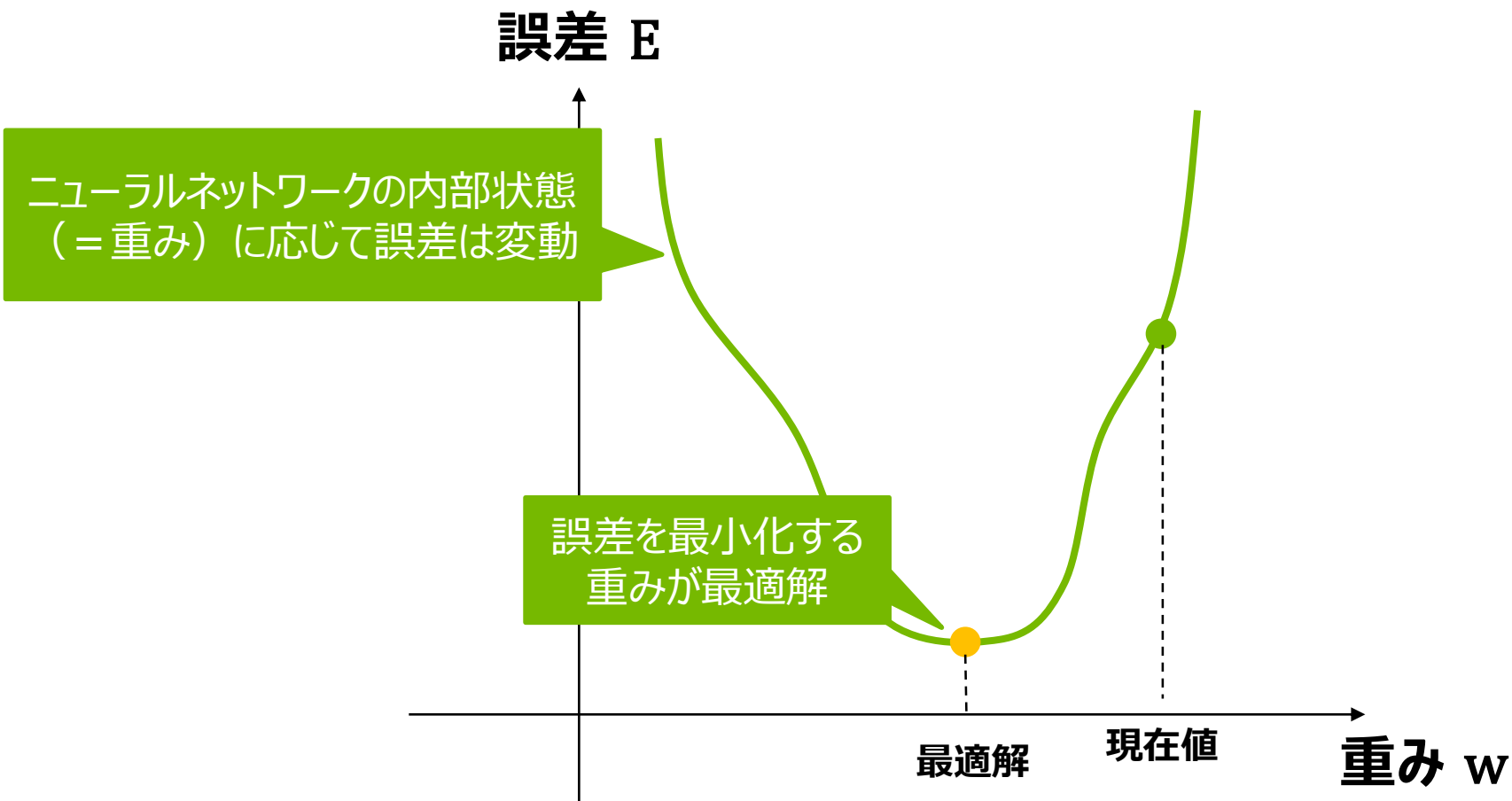
$z_1(\mathbf{x})$: 入力 tiger と予測した確率 $z_2(\mathbf{x})$: 入力 lion と予測した確率

ディープラーニングの学習フロー



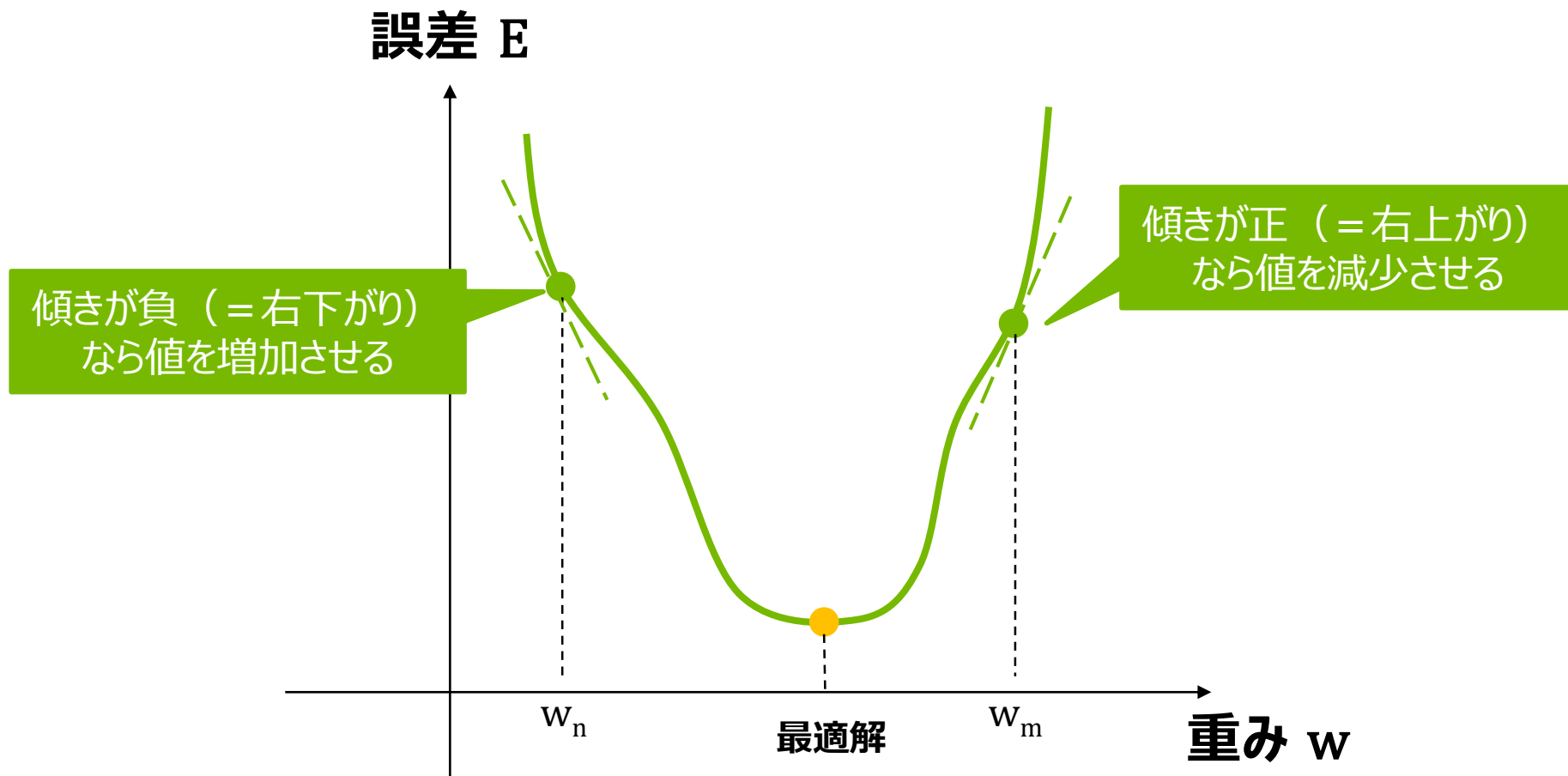
重み更新

勾配降下法による更新



重み更新

勾配降下法による更新



重み更新

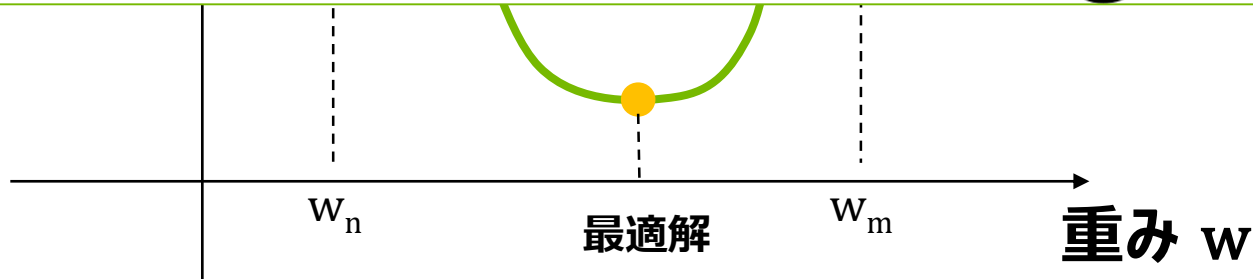
勾配降下法による更新

誤差 E

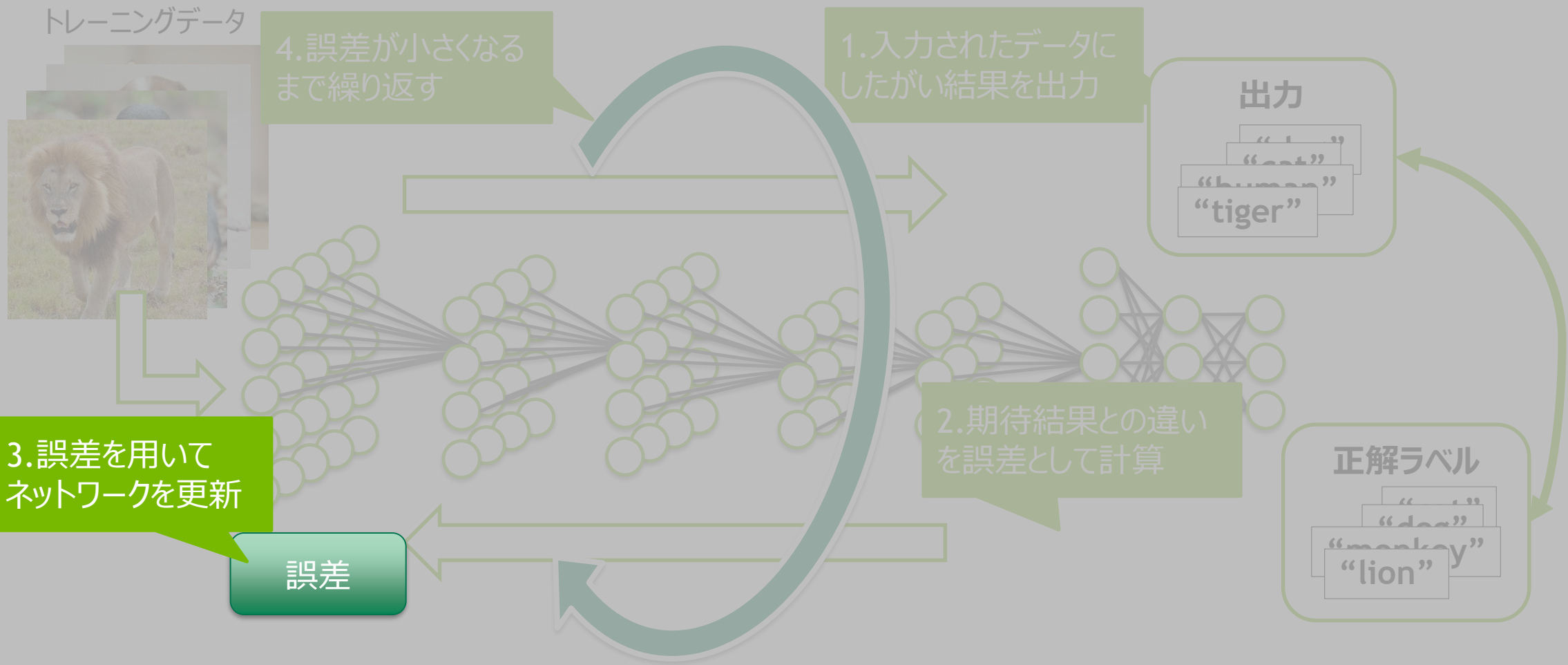
勾配ベクトルにより
重みを逐次更新

傾きが
なら

$$\mathbf{W}' = \mathbf{W} - \epsilon \frac{\partial E}{\partial \mathbf{W}}$$

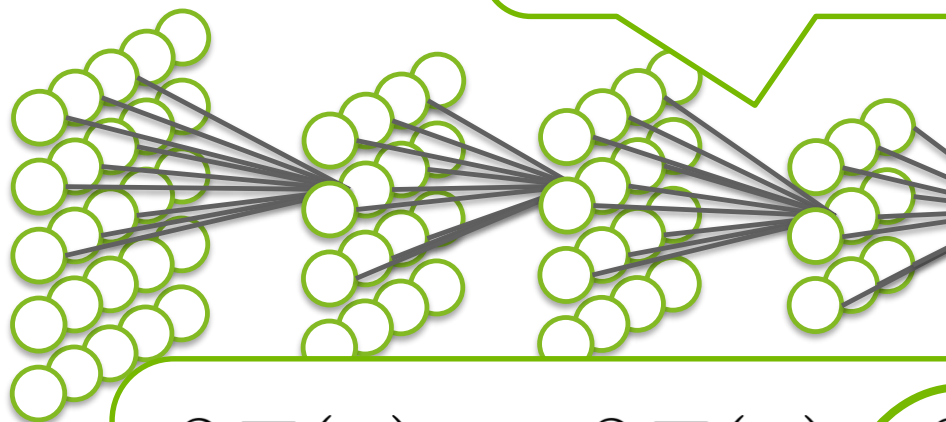


ディープラーニングの学習フロー



誤差逆伝播法

$w_{ji}^{(l)}$ を更新したい



直接計算することが
容易ではない

$$\frac{\partial E(\mathbf{z})}{\partial w_{ji}^{(l)}} = \frac{\partial E(\mathbf{z})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial w_{ji}^{(l)}} \text{が必要}$$

誤差逆伝播法

$$\frac{\partial E(\mathbf{z})}{\partial w_{ji}^{(l)}} = \frac{\partial E(\mathbf{z})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial w_{ji}^{(l)}} \quad \text{を直接考えるのではなく、}$$

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial w_{ji}^{(l)}} \quad \text{を考える}$$

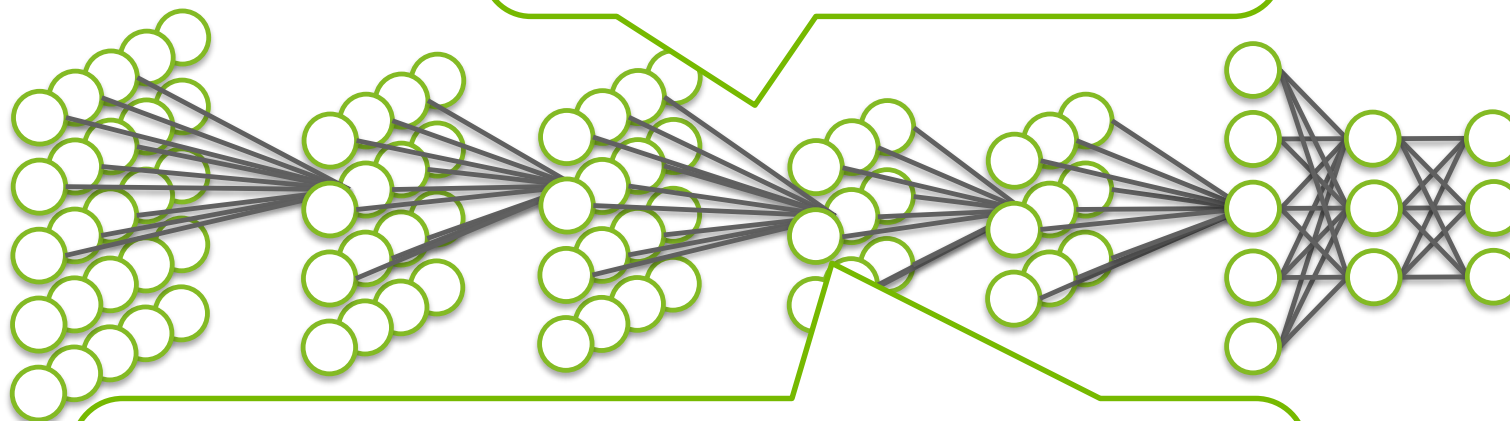
誤差逆伝播法

$$w_{ji}^{(l)} \text{ は } y_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)} \text{ としてのみ出現するので}$$

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial y_j^{(l)}} z_i^{(l-1)} \text{ とできる}$$

誤差逆伝播法

$w_{ji}^{(l)}$ を更新したい



$y_j^{(l)}$ は次の層の複数要素に影響

誤差逆伝播法

いろいろ計算すると、

$$\frac{\partial E}{\partial y_j^{(l)}} = \sum_k \frac{\partial E}{\partial y_k^{(l+1)}} \frac{\partial y_k^{(l+1)}}{\partial y_j^{(l)}}$$

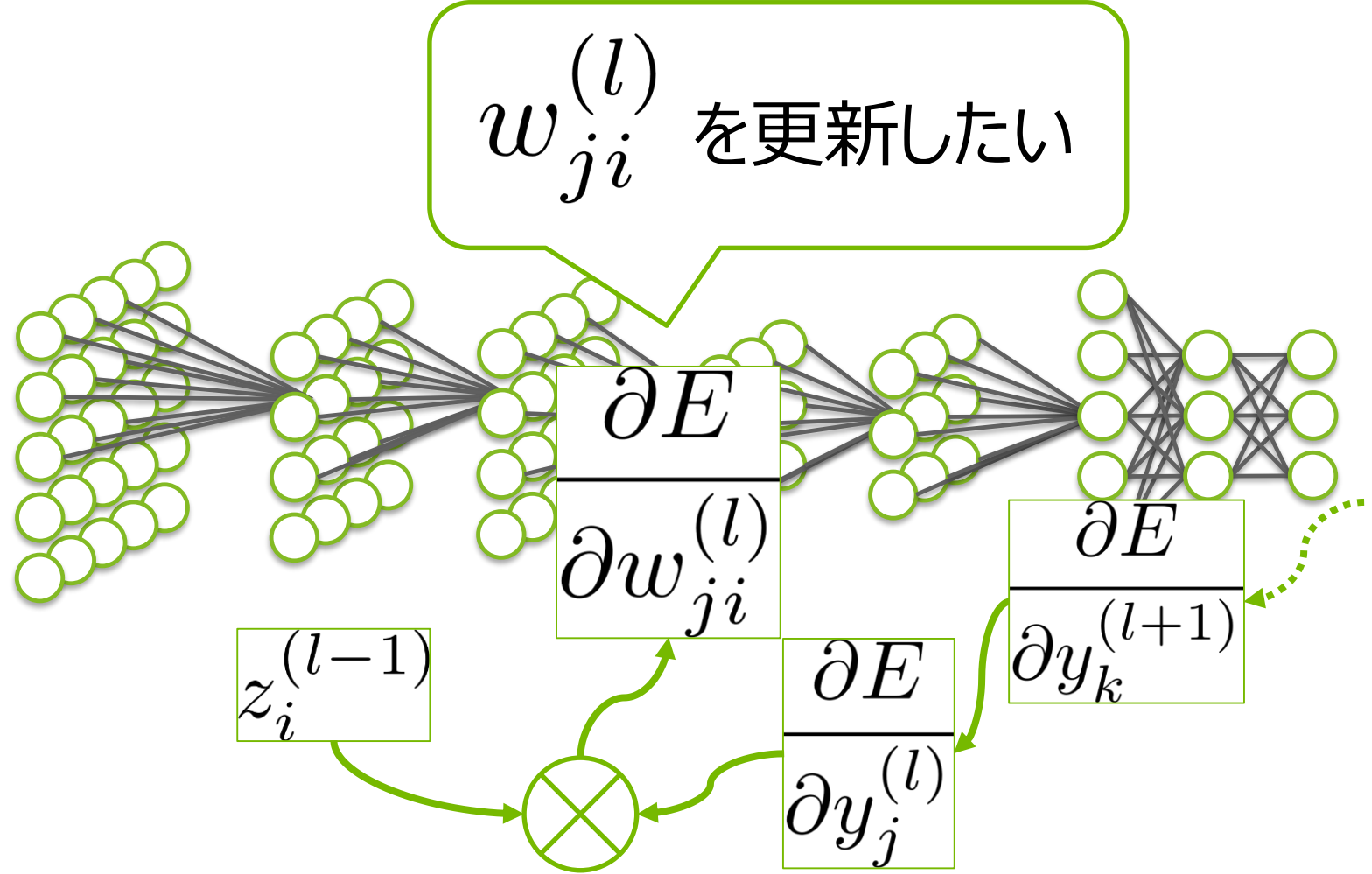
$$= \sum_k \left(\frac{\partial E}{\partial y_k^{(l+1)}} \left(w_{kj}^{(l+1)} F'(y_j^{(l)}) \right) \right) \text{となる。}$$

処理対象層の
情報なので計算可能

繰り返し処理により
計算可能

誤差逆伝播法

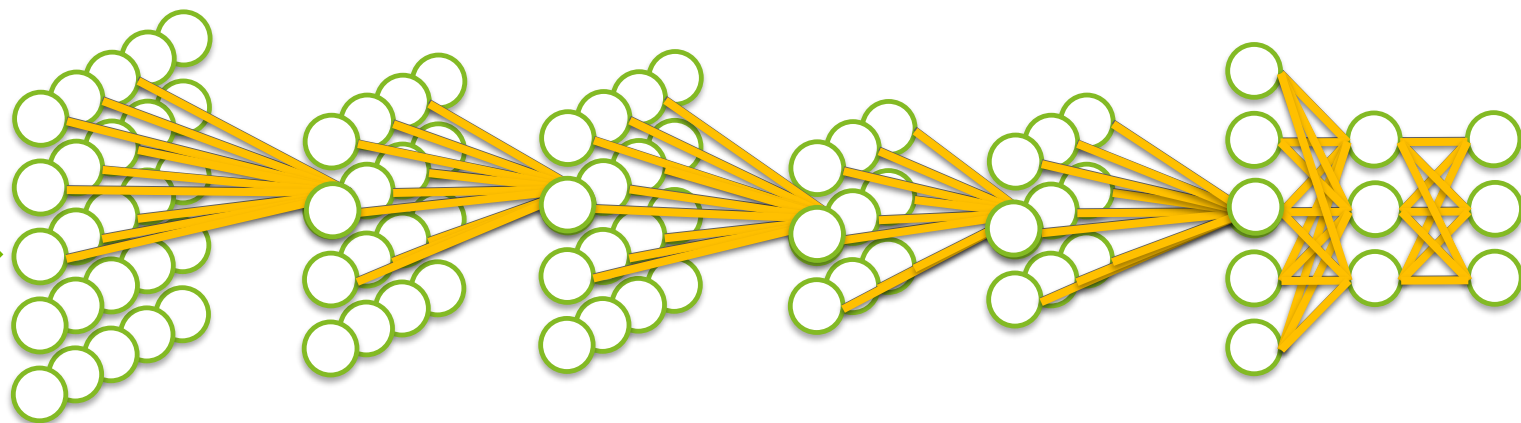
$w_{ji}^{(l)}$ を更新したい



誤差逆伝播法

処理のイメージ

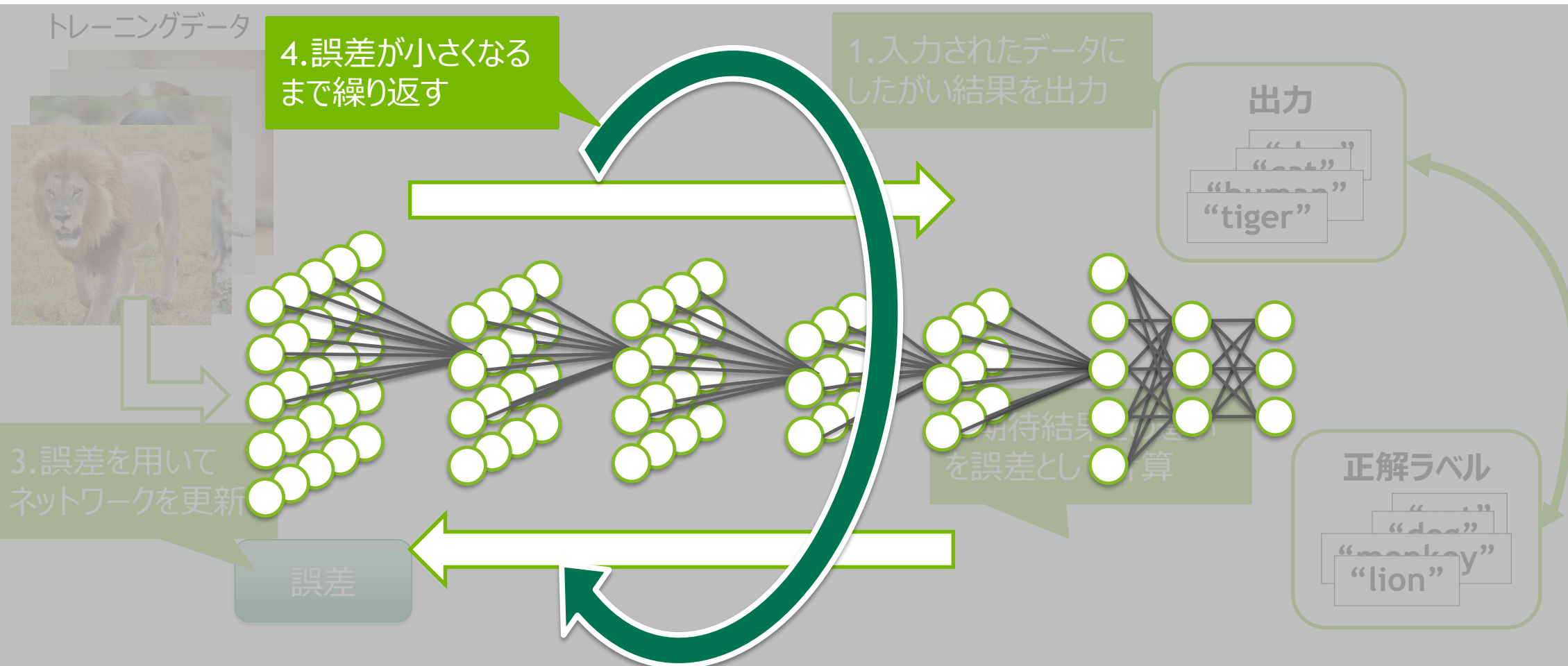
トレーニングデータ



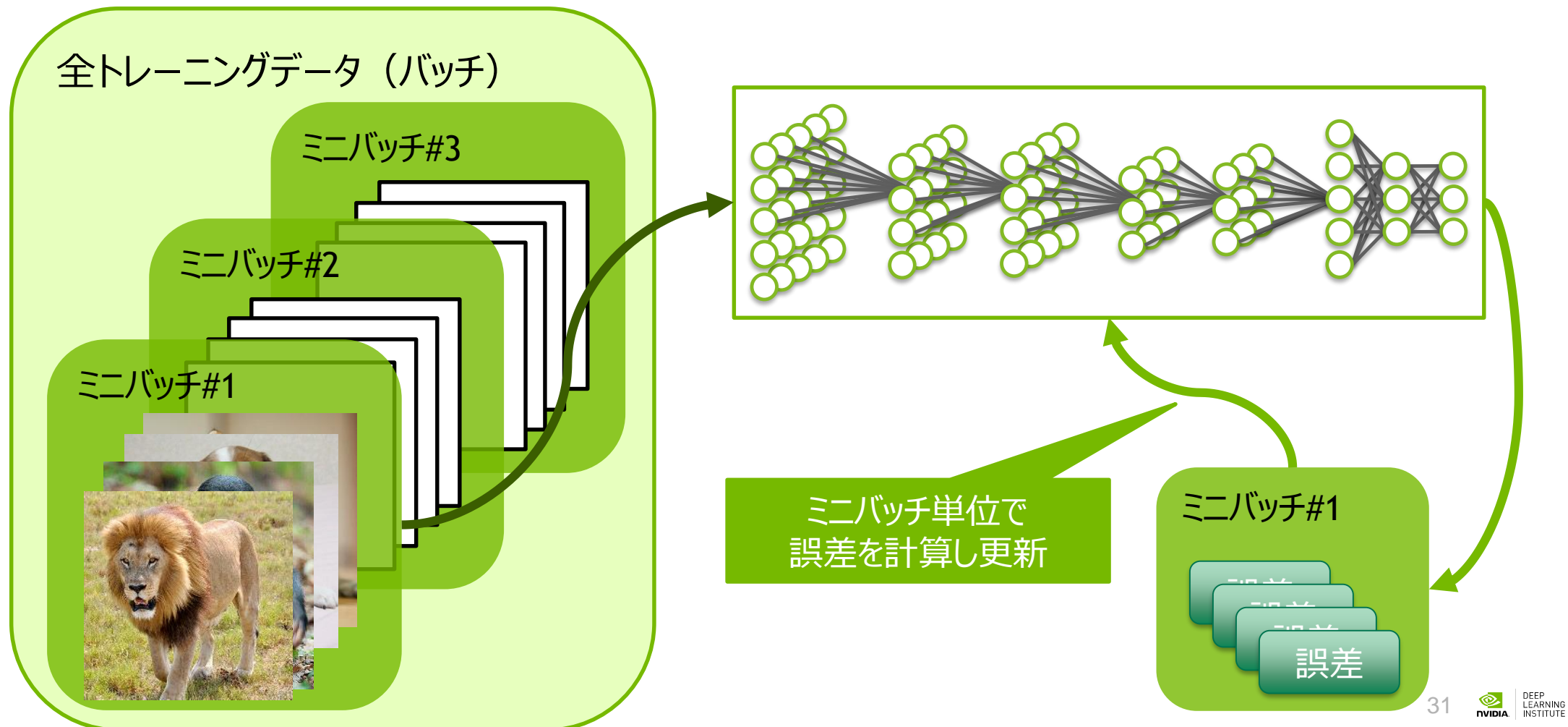
誤差

誤差を逆方向に伝えて、各層の重み行列を更新する

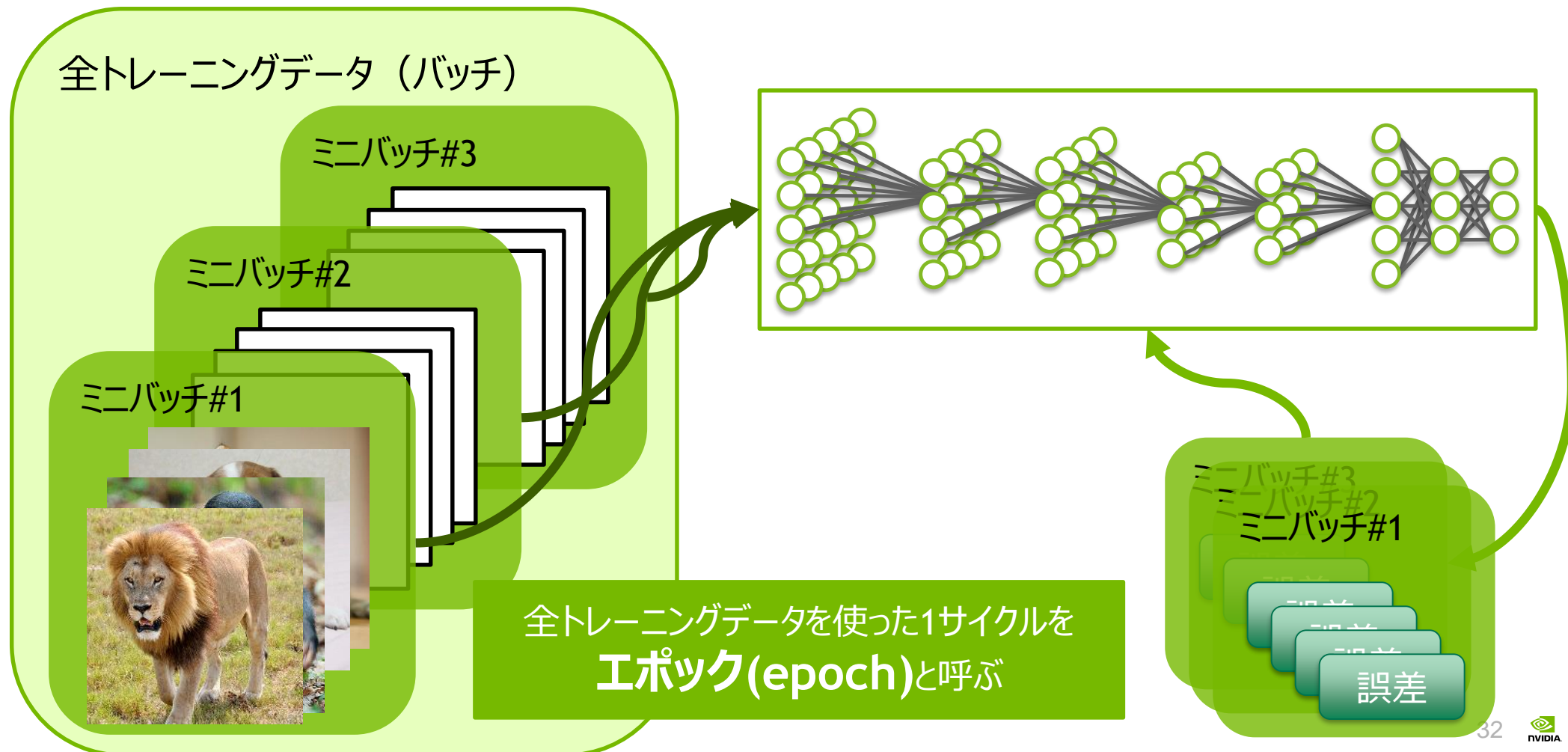
ディープラーニングの学習フロー



学習プロセスの単位: ミニバッチ



学習プロセスの単位: エポック



ディープラーニングを支えるソフトウェア

ディープラーニング フレームワーク

GPUで高速化されたディープラーニング フレームワークが多数存在

Caffe

DL4J
Deeplearning4j

Microsoft
CNTK

mxnet

Purine

torch

Caffe2

TensorFlow

theano

Chainer

PYTORCH

K
KERAS

Mocha.jl
julia

MatConvNet

MINERVA

OpenDeep

Pylearn2

ディープラーニング フレームワーク

NVIDIA DEEP LEARNING プラットフォーム

アプリケーション

Image Classification

Voice Recognition Translation

SPEECH AND AUDIO

Recommendation Engines Sentiment Analysis

BEHAVIOR

本日使用する
フレームワーク

フレームワーク

Caffe Chainer MxNet PyTorch TensorFlow theano

ディープラーニング
SDK

cuDNN TensorRT

DEEP LEARNING

cuBLAS cuSPARSE cuFFT

MATH LIBRARIES

GPU0 GPU1 GPU2 GPU3

NCCL

MULTI-GPU

GPU プラットフォーム

amazon web services IBM SOFTLAYER

Microsoft Azure 阿里云计算 Alibaba Cloud Computing

CLOUD

Tesla V100 Tesla P40/P4 Jetson TX2 DRIVE PX2

GPU

DGX-1

SERVER



本日使用するフレームワーク: CHAINER&CHAINERMN

CHAINER

Pythonベースのディープラーニング フレームワーク

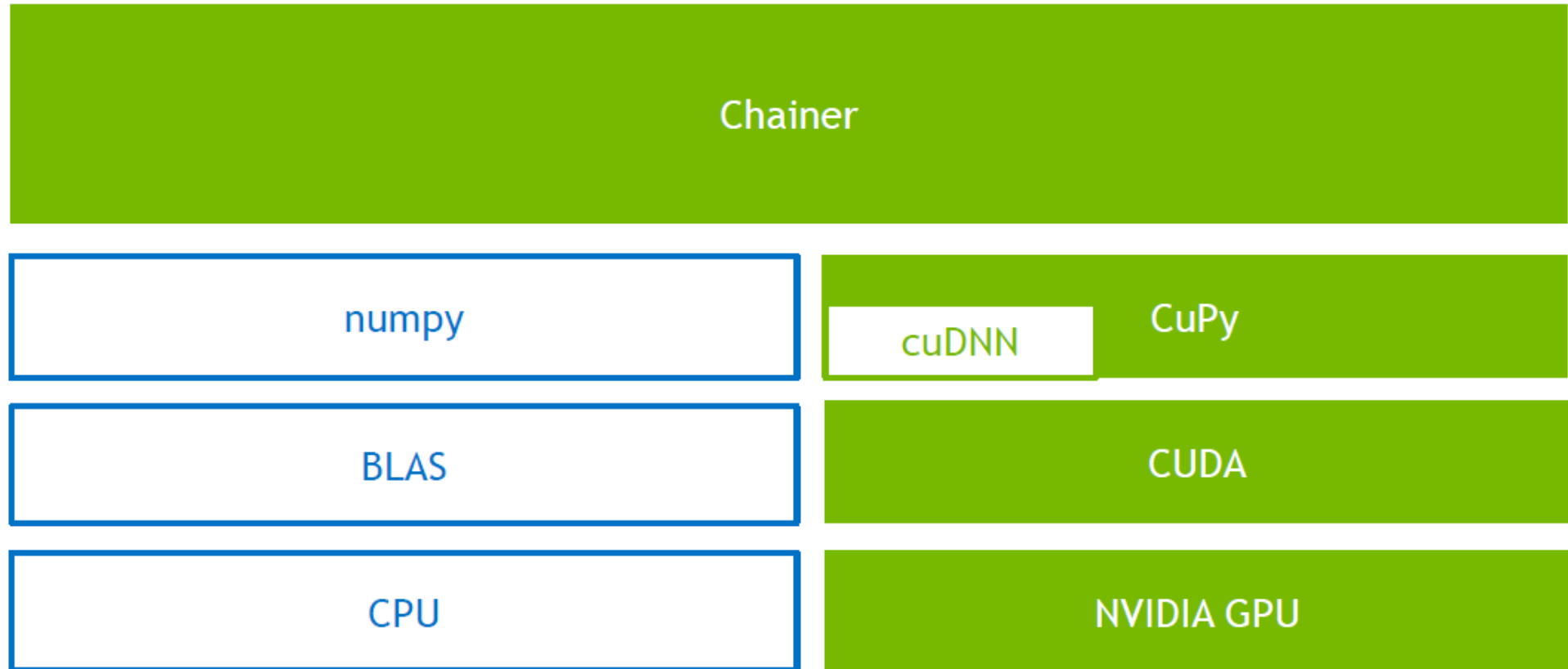
- Preferred Networks社によって開発
 - 最新はv4.1.0 (2018/05/25時点)
- NumPyインターフェース互換のGPU用数値計算ライブラリCuPyで計算を高速化
- 柔軟に記述でき、デバッグしやすく、直感的
- Define-by-Run方式(後述)の先駆者



<https://chainer.org/>
<https://github.com/chainer/chainer>

CHAINER

ソフトウェアスタックのイメージ



CHAINER

特徴的な実行方式: Define-by-Run

```
# Define-and-Run
# 計算グラフを構築後に、計算を実行

x = Variable('x')
w = Variable('w')
y = x * w

for xi, wi in data:
    eval(y, (xi, wi))
```

```
# Define-by-Run
# 計算グラフの構築と実行が同時

for xi, wi in data:
    x = Variable(xi)
    w = Variable(wi)
    y = x * w
```

データに応じて処理を
分岐することが容易

CHAINER

特徴的な実行方式: Define-by-Run

```
# Define-and-Run
# 計算グラフを構築後に、計算を実行

x = Variable('x')
w = Variable('w')
y = x * w

for xi, wi in data:
    eval(y, (xi, wi))
```

```
# Define-by-Run
# 計算グラフの構築と実行が同時

for i, (xi, wi) in enumerate(data):
    x = Variable(xi)
    w = Variable(wi)
    if i % 2 == 0:
        y = x * w
    else:
        y = x + w
```


CUPY

GPUで動作するNumPy互換の数値計算ライブラリ

- CUDAで高速化された数値計算ライブラリ
 - 最新はv4.1.0 (2018/05/25時点)
- NumPy互換のI/Fを持つ
 - CPUで実装したコードをシームレスにGPU化
- もとはChainerの内部モジュールとして開発
 - 現在は単独でも提供されている
 - インストールコマンドは `pip install cupy` など



<https://cupy.chainer.org/>
<https://github.com/cupy/cupy>

CUPY

GPUで動作するNumPy互換の数値計算ライブラリ

NumPyコード

```
import numpy

x = numpy.array([1,2,3], numpy.float32)
y = x * x
s = numpy.sum(y)
print(s)
```

CuPyコード

```
import cupy

x = cupy.array([1,2,3], cupy.float32)
y = x * x
s = cupy.sum(y)
print(s)
```

CHAINERMN

Chainerをベースとした分散学習用ライブラリ

- マルチノードでの分散学習をサポートするための追加パッケージ

- 最新はv1.3.0 (2018/05/25時点)

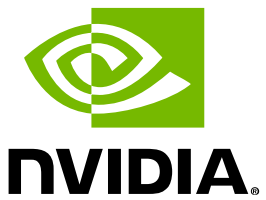
- CUDA-aware MPIやNCCLなどのライブラリを利用して実装
- 大規模なクラスタでも非常によくスケールする



<https://github.com/chainer/chainermn>

まとめ

- ディープラーニングはニューラルネットワークを基本としている
- 学習では、勾配法や誤差逆伝播法を用いて、パラメータを更新
- さまざまなフレームワークがあり、用途に応じて使い分けることも重要



DEEP
LEARNING
INSTITUTE

www.nvidia.com/dli