2015/12/25
自動チューニング研究会
@山上会館

ディープラーニングフレームワークChainerの紹介と自動チューニングへの期待

株式会社 Preferred Networks
大野健太 oono@preferred.jp

自己紹介

- 大野健太(@delta2323_)
  - 経歴：数理科学研究科・修士課程（共形幾何）
    - → 2012.3 PFI → 2014.10 PFN
  - 所属：研究班（理論解析・ライフサイエンス・Chainer開発メンバー）
  - ブログ：http://delta2323.github.io
- 最近の活動
  - NIPS2014勉強会・ICML2015勉強会主催
  - 日経ビッグデータ短期連載・雑誌寄稿など
  - **NIPS2015読み会1月下旬に開催（予定）**

Preferred Infrastructure

株式会社Preferred Networks

代表取締役
西川徹

取締役副社長
岡野原大輔

Preferred Infrastructure（PFI、2006年-）

- 検索・機械学習のソフトウェア研究開発

Preferred Networks（PFN、2014年にPFIよりスピンアウト）

- IoT時代の機械学習・深層学習技術の開発
- 出資
  - 2014年10月 日本電信電話（持ち株会社）（2億円）
  - 2015年8月 FANUC（9億円）
  - 2015年10月 トヨタ自動車（10億円）
- 協業・共同研究
  - 日本電信電話、FANUC、トヨタ自動車、Panasonic、NVIDIA、CiRA

アジェンダ

- 深層学習概要
- 深層学習基礎
- 深層学習フレームワークChainer
- 深層学習フレームワークの類別
- 深層学習の最適化の課題（講演では未紹介）

Preferred Infrastructure.

深層学習概要

Preferred Infrastructure.

機械学習とは

“Machine learning is the science of getting computers to act without being explicitly programmed.” (Andrew Ng, Coursera, Machine Larningの「About this course」より)

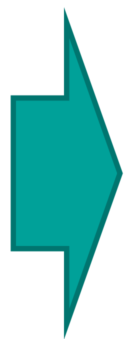
データを用いて機械に対して高度な判断を行わせる手法

機械に人手で明示的にルールを与えるのではなく、機械にデータを与え、データ内の法則・傾向・パターンなどの知見を機械自身に抽出させる

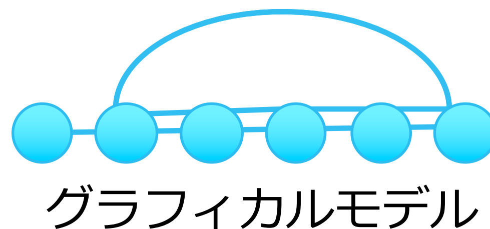
タスク	入力	出力
メール分類	メール	スパム or 普通 or 重要等
Twitterのユーザー分析	Tweet	ユーザーの性別、職業、年齢など
電気使用料需要の予測	パケット	各サーバーの予測使用量（連続値）
広告のコンバージョン予測	アクセス履歴、広告	クリック、コンバージョンするか
監視カメラ解析	監視カメラ画像	部屋の状態（明かりがついている？人がいるか？など）

機械学習の典型的なプロセス

画像　センサー



(0,　1, 2.5, -1, …)
(1, 0.5,　-2,　3, …)
(0,　1, 1.5,　2, …)
特徴ベクトル



分類/回帰
SVM/LogReg/PA
CW/ALOW/Naïve
Bayes/CNB/DT
RF/ANN…

クラスタリング
K-means/Spectral
Clustering/MMC/L
SI/LDA/GM…

構造分析
HMM/MRF/CRF…

様々な様式の
生データ

分野に依存しない
抽象化データ

様々な手法
理論を適用

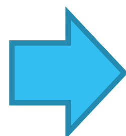
Infrastructure.

特徴抽出

例：固有名詞を取り出してニュース記事の特徴とする

２０２０年の東京五輪・パラリンピックの主会場となる新国立競技場をめぐり、安倍晋三首相は、総工費が２５２０億円に膨らんだ建設計画を見直す考えを１７日に表明する方向で最終調整に入った。競技場を１９年のラグビーワールドカップ（W杯）の主会場にする計画は断念する。同日、東京五輪・パラリンピック組織委員会会長の森喜朗元首相と会談し、計画見直しへの協力を求める方針だ。

２０２０年の**東京五輪**・**パラリンピック**の主会場となる**新国立競技場**をめぐり、**安倍晋三首相**は、総工費が２５２０億円に膨らんだ建設計画を見直す考えを１７日に表明する方向で最終調整に入った。競技場を１９年の**ラグビーワールドカップ**（**W杯**）の主会場にする計画は断念する。同日、**東京五輪**・**パラリンピック**組織委員会会長の**森喜朗元首相**と会談し、計画見直しへの協力を求める方針だ。



単語	頻度
東京五輪	2
パラリンピック	2
新国立競技場	1
安倍晋三	1
・・・	・・・

機械学習アルゴリズム

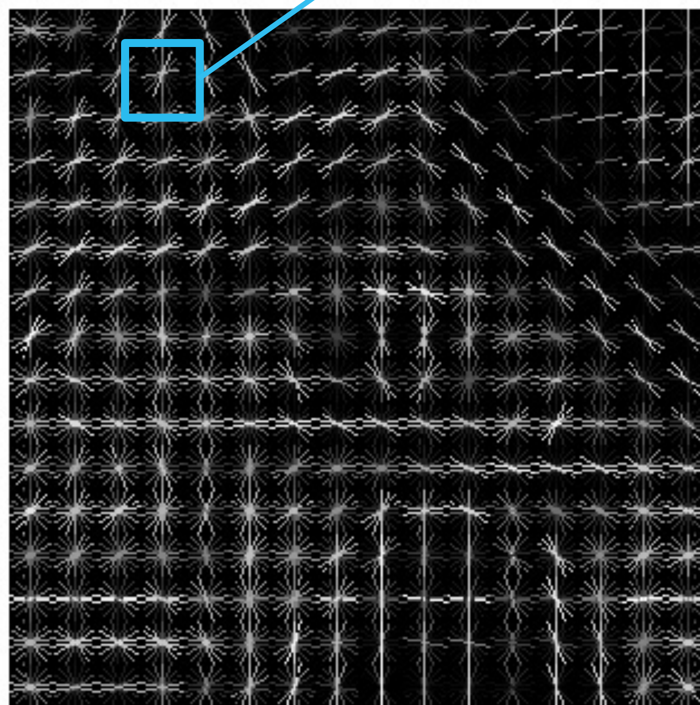
ニュース記事：
http://www.asahi.com/articles/ASH7J7GKMH7JULFA038.html

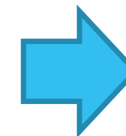
Preferred Infrastructure

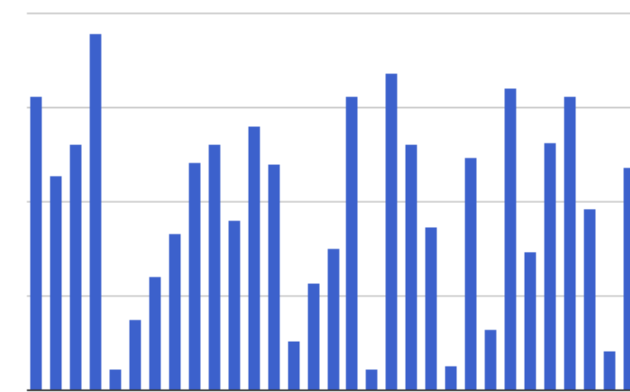
画像からの特徴抽出

例：Histogram of Gradient（HoG特徴量）

各ピクセルでの勾配を小ブロック（セル）単位でまとめてヒストグラム化







各セルでのヒストグラムを（正規化して）すべてまとめる

機械学習
アルゴリズム

http://www.vlfeat.org/overview/hog.htmlより

特徴抽出は職人技

- 特徴抽出の重要性
  - 特徴の良し悪しが学習精度に大きく影響
  - 学習アルゴリズムの選択以上に精度に効く場合も
- 特徴抽出は難しい
  - タスクごとに最適な特徴抽出方法は異なる
  - 機械学習コンテストは最後は特徴抽出のチューニング勝負
- これまで様々な特徴抽出方法が研究されてきた
  - 自然言語：n-gram/BoW　画像：SIFT/SURF/HOG/PHOW/BoVW
  - その他にも様々なヒューリスティックが存在

Preferred Infrastructure

特徴抽出

例：固有名詞を取り出してニュース記事の特徴とする

２０２０年の東京五輪・パラリンピックの主会場となる新国立競技場をめぐり、安倍晋三首相は、総工費が２５２０億円に膨らんだ建設計画を見直す考えを１７日に表明する方向で最終調整に入った。競技場を１９年のラグビーワールドカップ（W杯）の主会場にする計画は断念する。同日、東京五輪・パラリンピック組織委員会会長の森喜朗元首相と会談し、計画見直しへの協力を求める方針だ。

ニューラルネットワーク

Preferred Infrastructure.

2012年画像認識コンテストで Deep Learningを用いたチームが優勝

ILSVRC2012
優勝チームSupervisonの結果
[Krizhevsky+12] ↓

衝撃的な出来事

- 限界と思われた認識エラーを4割も減らした (26%→16%)
- 特徴抽出を行わず、生の画素をNNに与えた

第3次ニューラルネットブーム

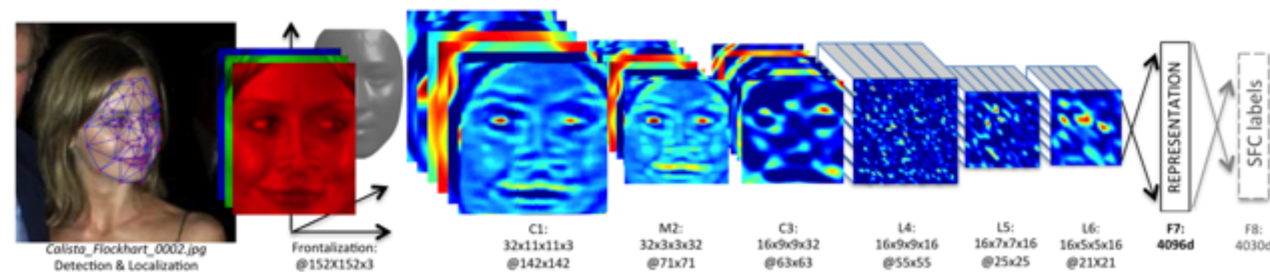
- 翌年の同コンテストの上位陣はほぼディープラーニングベースの手法
- その後もエラー率低減の競争が続く
  - 16%(‘12) → 11%(‘13) → 6.6%(‘14) → 3.9%(‘15)

Neural Netブーム

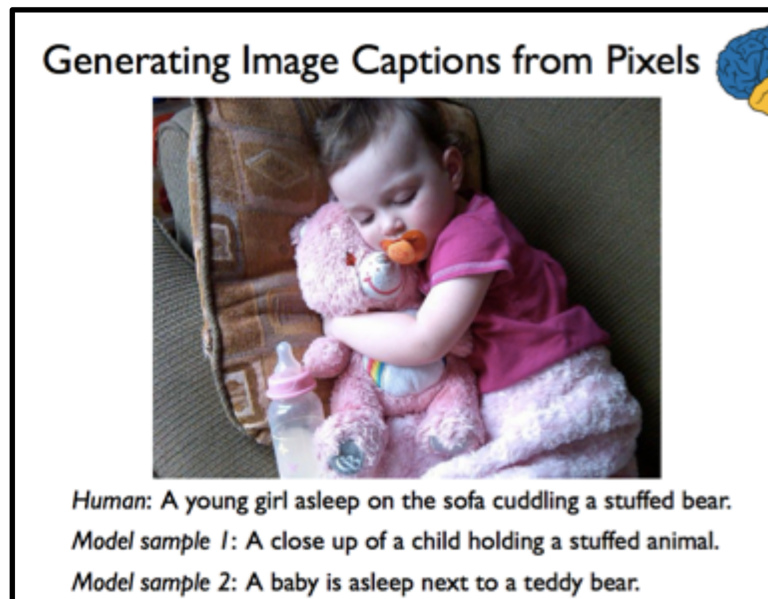
- 各企業がDL研究者の獲得競争
  - Google/Facebook/Microsoft/Baidu
- 実サービスもDLベースに置き換えられる
  - Siri/Google Photos/YJVOICE

←Google Brainによる猫認識 [Le, Ng, Jeffrey+12]

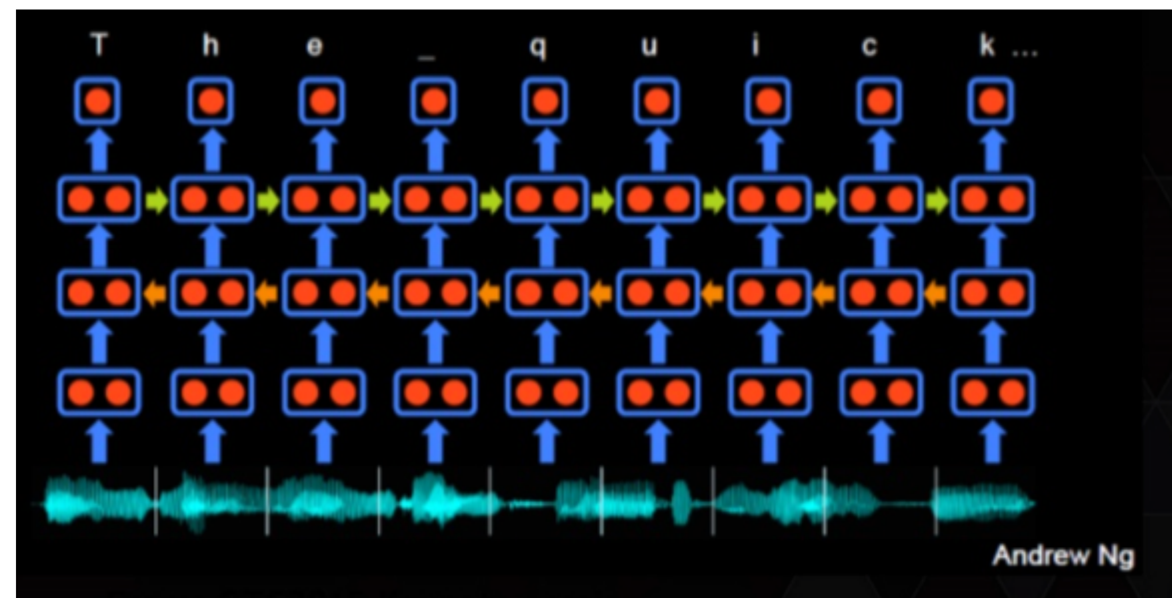
http://research.google.com/archive/unsupervised_icml2012.html



↑DeepFace[Taigman+'14]↓DeepSpeech[Hannun+'14]*



キャプション生成↑
Show and Tell[Vinyal+14]



*http://www.slideshare.net/NVIDIAJapan/gpu-51812528

ニューラルネットワークが利用されたタスク

データ	画像				
タスク	カテゴリ分類	顔検出	生成	ゲームAI	シーン認識

	自然言語			音声	化合物
	表現学習	翻訳	質問応答	会話検出	QSAR (活性予測)

	動画		画像+ 自然言語		音声+動画
	カテゴリ分類	動作認識	キャプション生成	表現学習	音声認識

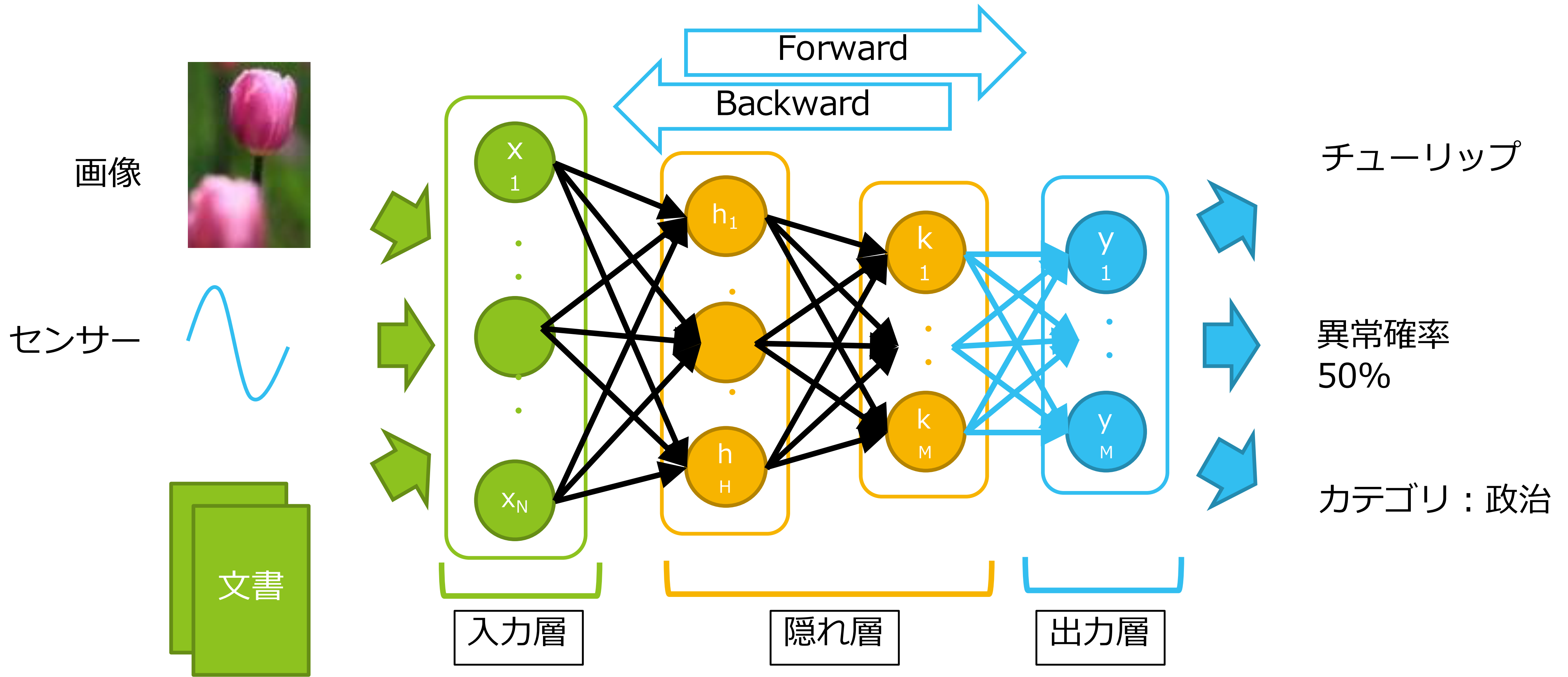
応用分野
音声検索 画像キュレーション eコマース 自動運転 ロボティックス 医療画像 マーケティング

Preferred Infrastructure.

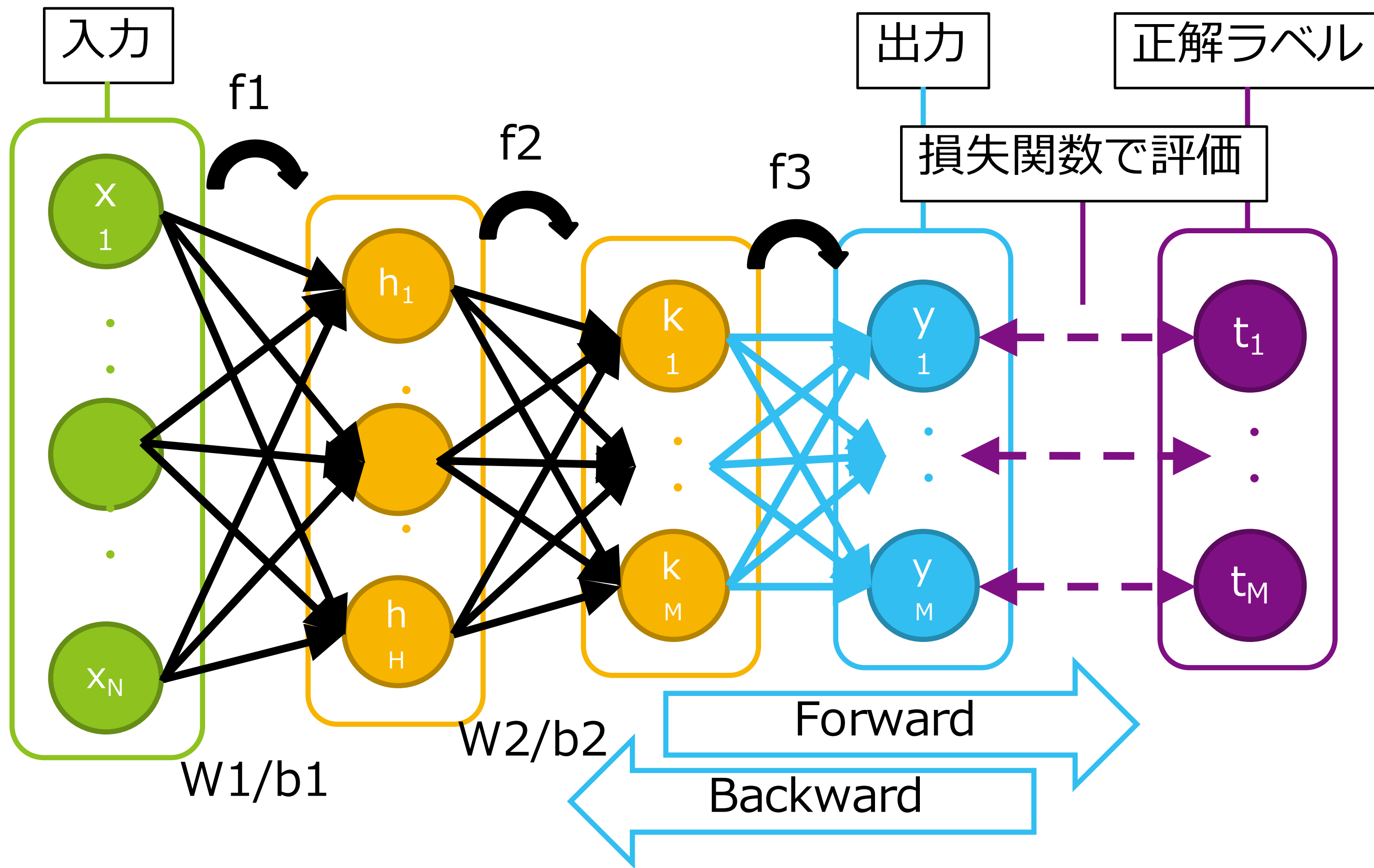
深層学習基礎

Preferred Infrastructure.

典型的なニューラルネットワーク（多層パーセプトロン）



多層パーセプトロン（Multi Layer Perceptron, MLP）



訓練データ
- 特徴ベクトル : x1, x2, …, xN
- 正解ラベル : t1, t2, …, tN

Forward更新式
- h = f1(x) = Sigmoid(W1x + b1)
- k = f2(h) = Sigmoid(W2h + b2)
- y = f3(k) = SoftMax(k)
  $y_i = \exp(k_i)/\Sigma_j \exp(k_j)$

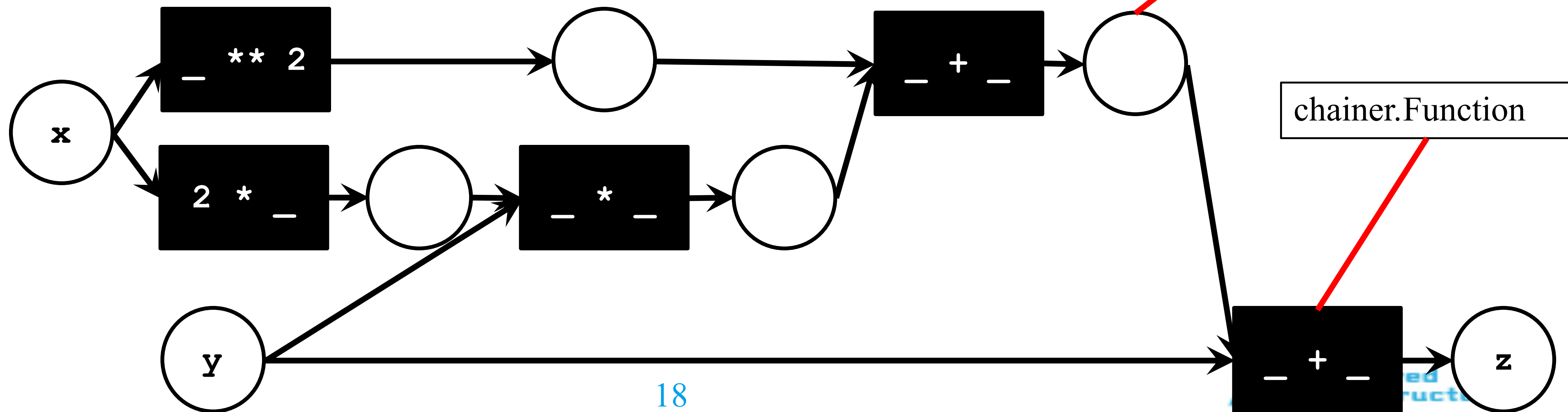
学習すべきパラメータ
- W1 : 1層目のパラメータ行列
- b1 : 1層目のバイアス項
- W2 : 2層目のパラメータ行列
- b2 : 2層目のバイアス項

損失関数の計算方法
- $Loss = \Sigma_n\, loss(yn, tn)$
  $loss(y, t) = \Sigma_m\, t_m \log(y_m)$

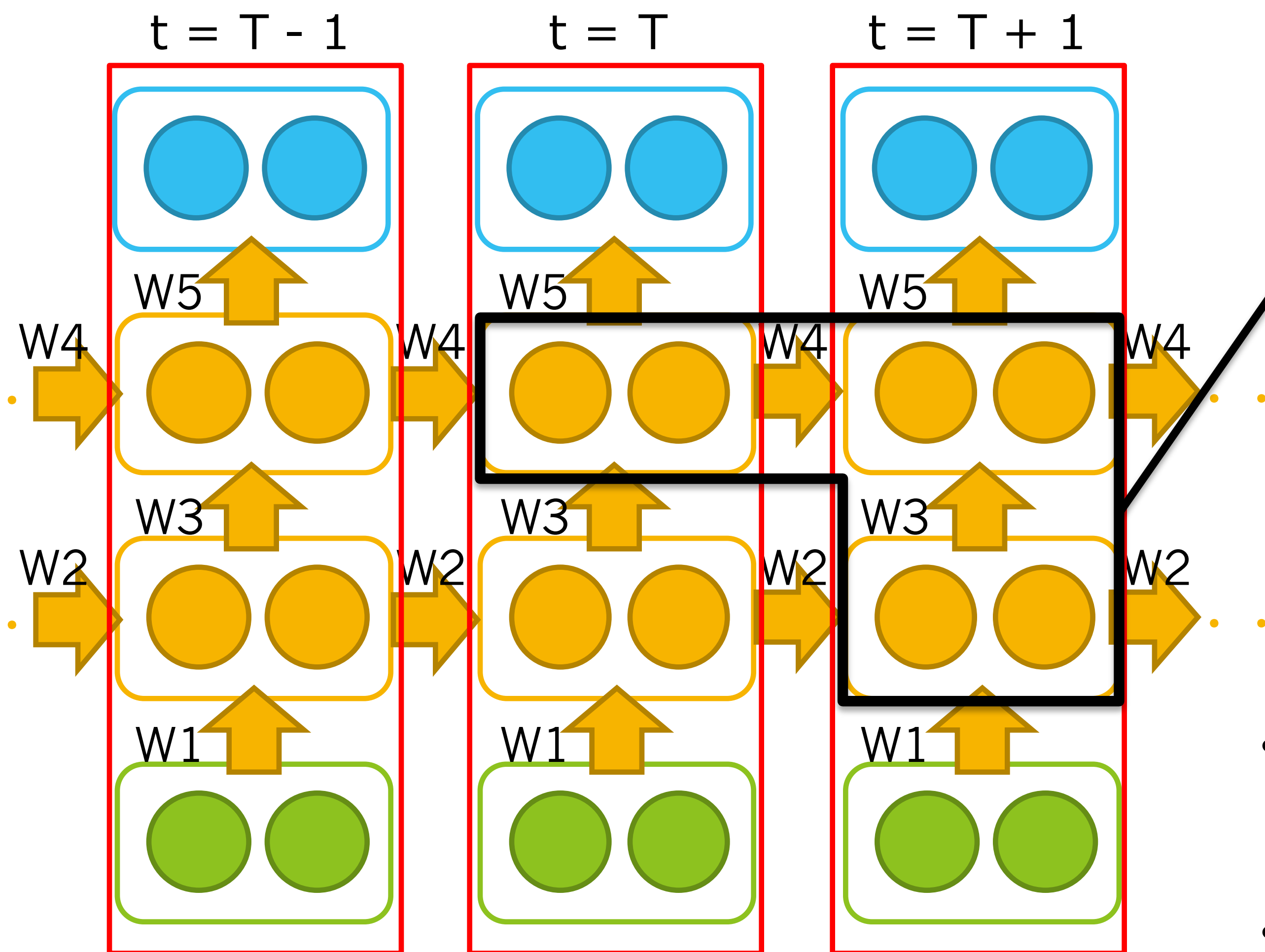
各層は「全結合層(W/b)」と「活性化関数(Sigmoid)」からなる

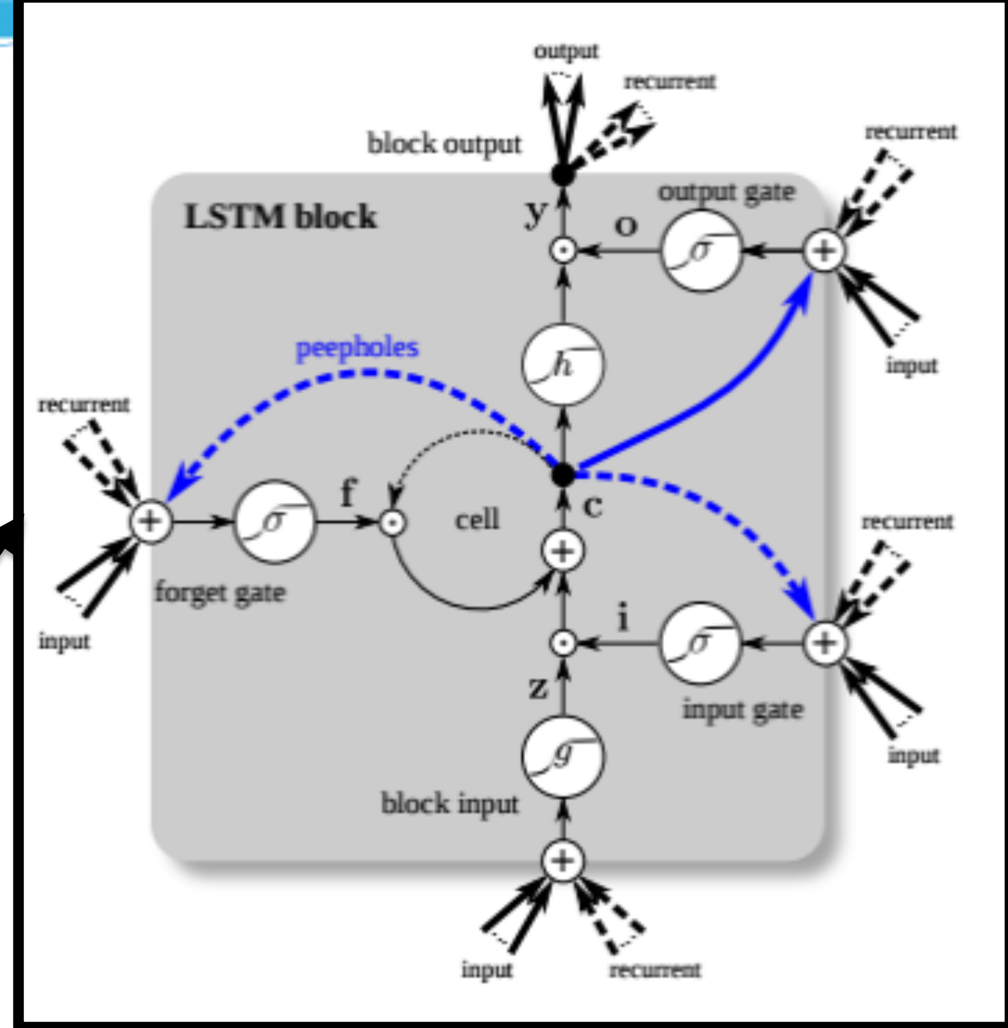
現代的なニューラルネットワーク = 計算グラフ

```
x = chainer.Variable(np.array(1))
y = chainer.Variable(np.array(1))
z = x**2 + 2*x*y + y
z.backward()
```



複雑なNNの例：
Long Short Term Memory (LSTM)

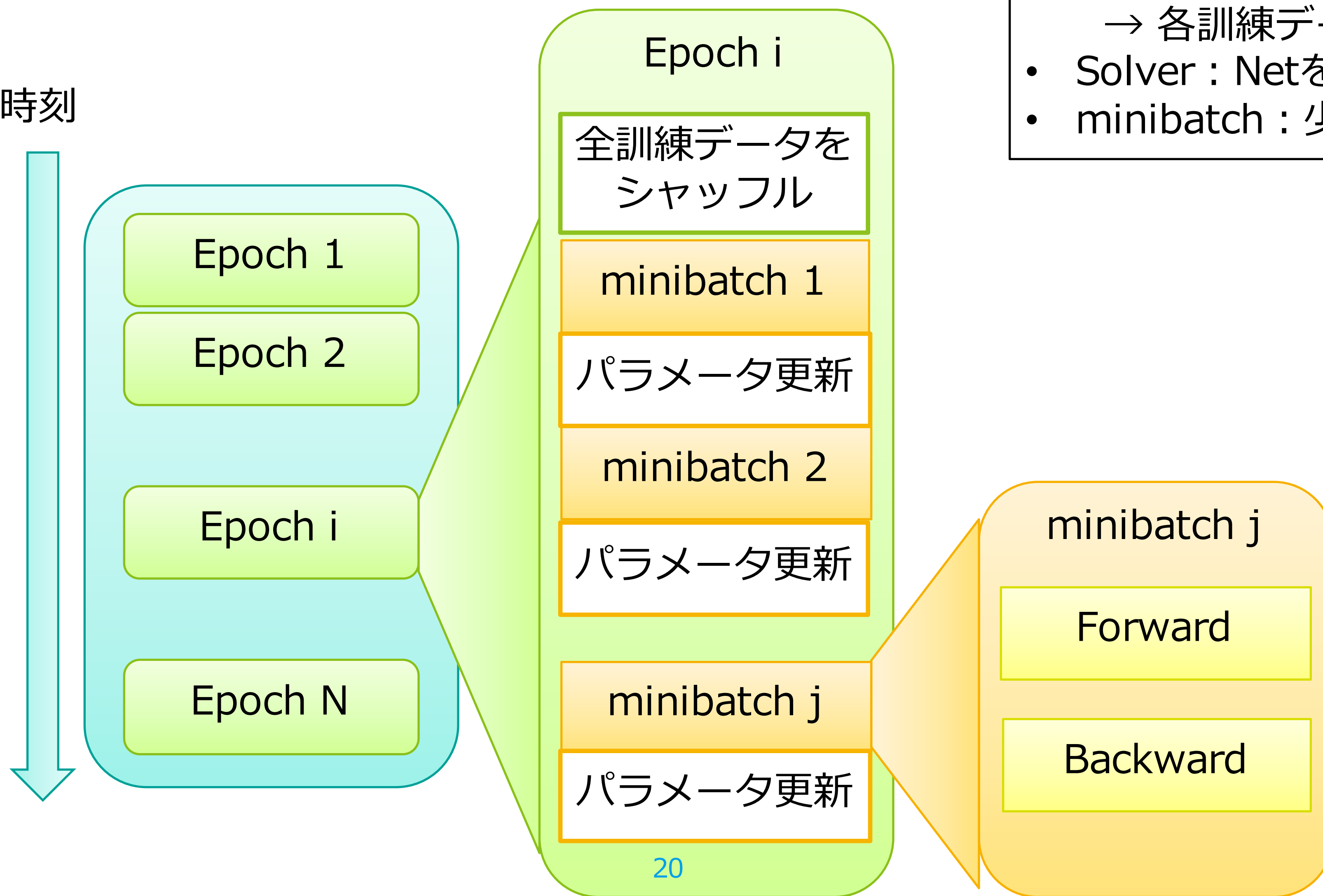




- 主に時系列データを扱うNNであるRecurrent Neural Network (RNN)の一種
- 長期間の依存関係を学習の困難さを解決に提案されたモデル

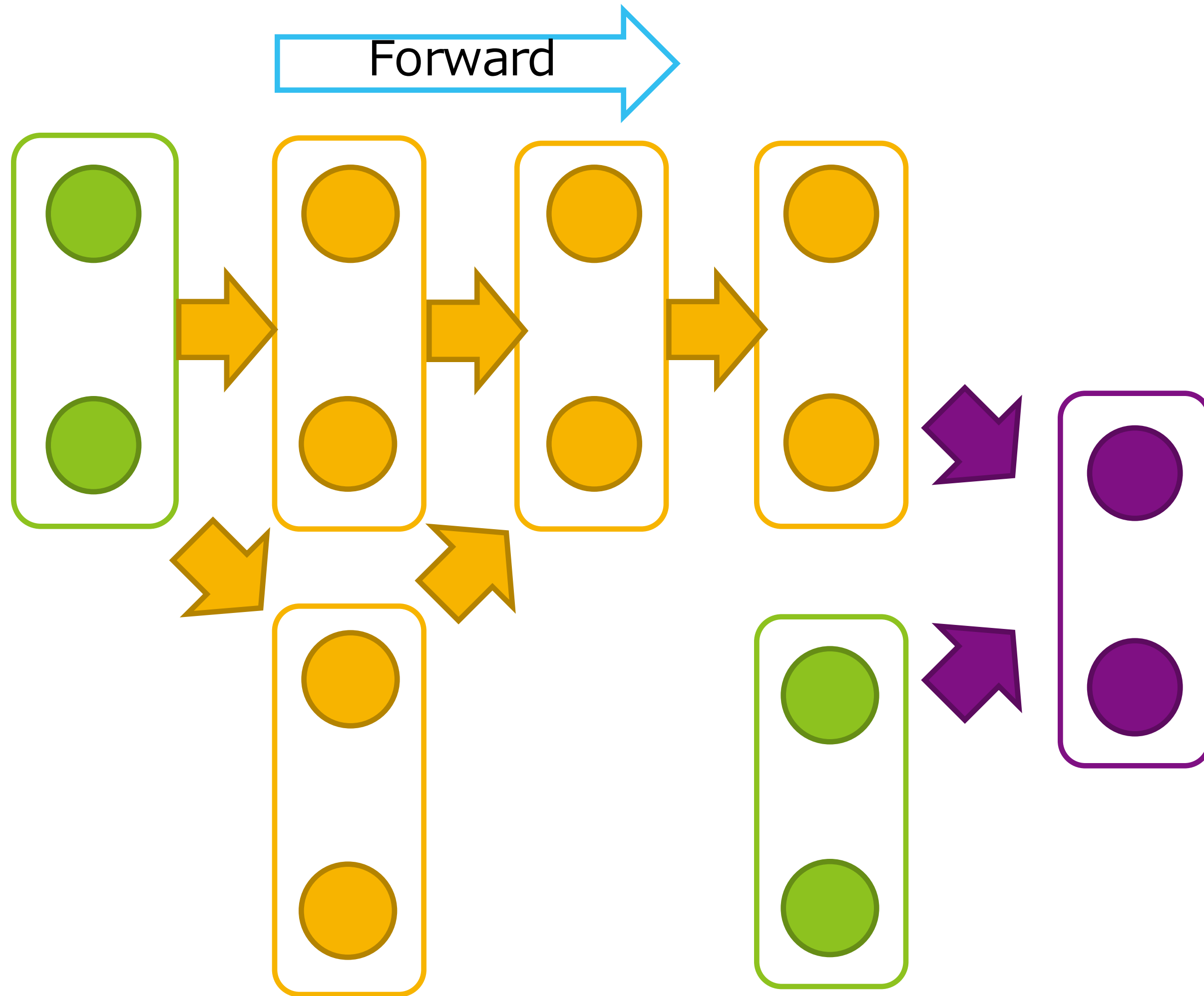
訓練の流れ

- Epoch (Iteration)：全訓練データを1巡する事
  → 各訓練データはNetにN回与える
- Solver：Netを訓練するモジュール
- minibatch：少数の訓練データをまとめたもの



Preferred Infrastructure.

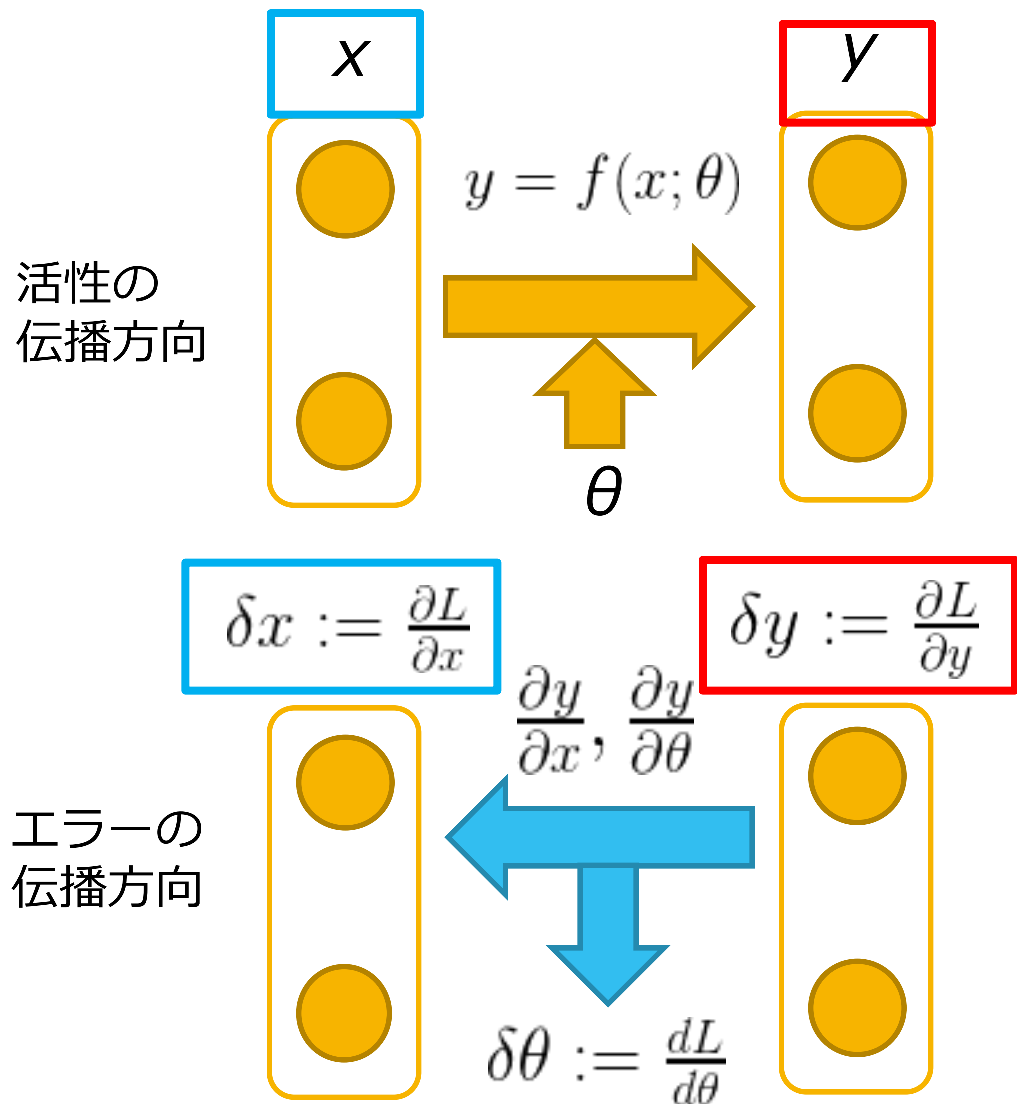
Forward Propagation （順伝播）



- 計算グラフの先頭のユニット（緑）に値を与え、順方向に計算を進める
- Forward計算の過程で損失(Loss) を計算する
  - 通常損失は計算グラフの最後のユニット（紫）での値を指す
- 損失は各ユニットの値や各レイヤーのパラメータの関数になっている

Preferred Infrastructure.

連鎖律（Chain Rule）



- Forward Propagationを以下のように書く

$$y = f(x;\theta)$$

  - （θ：Layerのパラメータ、全結合層の重み等）

- 損失をLとすると、連鎖律より

$$\frac{\partial L}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial L}{\partial y}, \quad \frac{\partial L}{\partial \theta} = \frac{\partial y}{\partial \theta}\frac{\partial L}{\partial y}$$

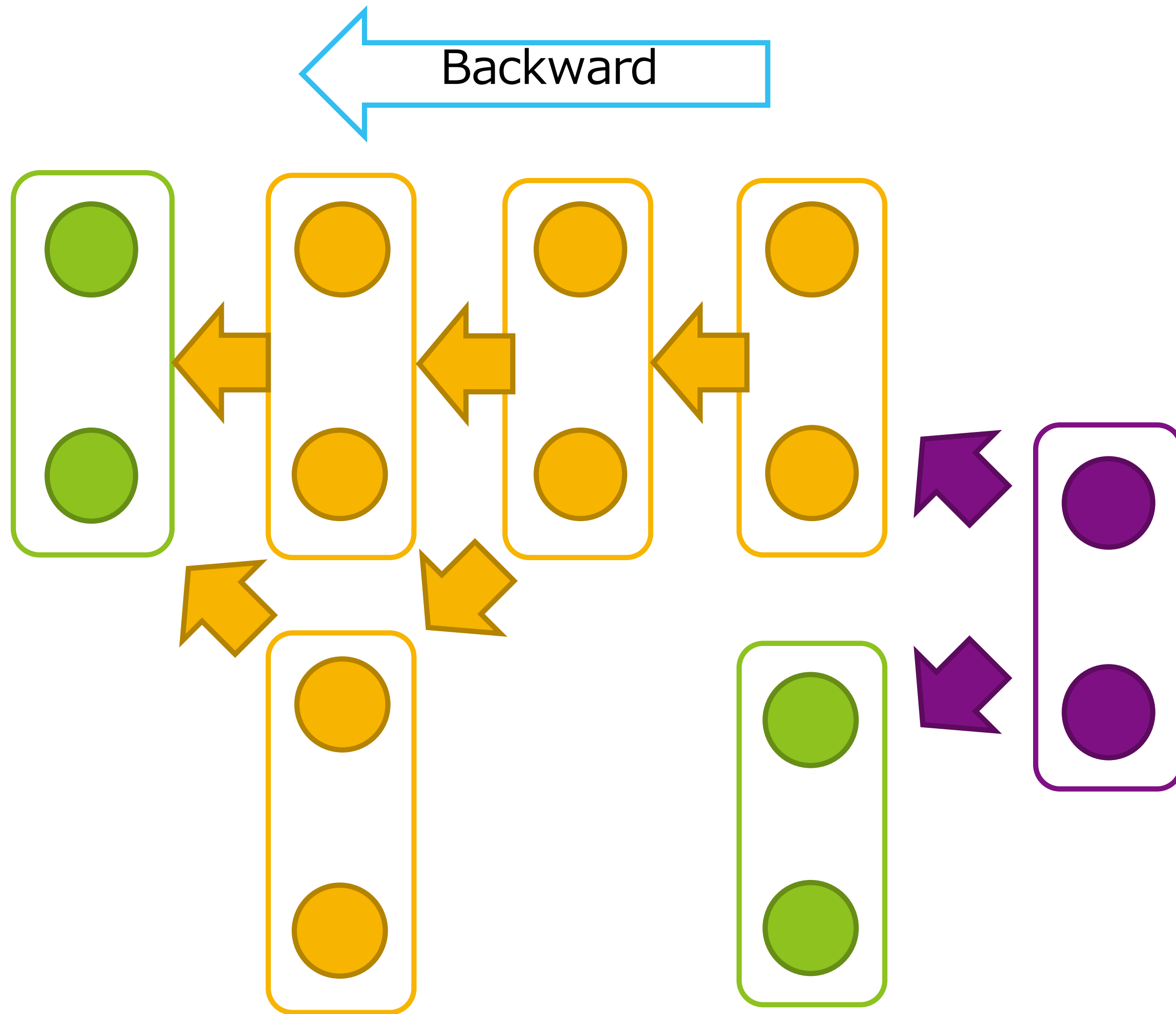
- 勾配を $\delta x := \frac{\partial L}{\partial x}$ $\delta\theta := \frac{\partial L}{\partial \theta}$ などと書くと

$$\delta x = \frac{\partial y}{\partial x}\delta y, \quad \delta\theta = \frac{\partial y}{\partial \theta}\delta y$$

→ **エラーは活性と逆向きに伝播する**

Preferred Infrastructure

Backward Propagation（逆伝播、誤差逆伝播）



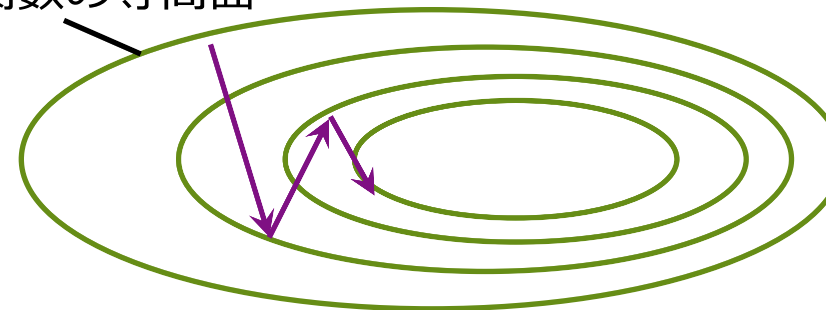
- 計算グラフの末端のユニット（紫）にエラーを与え、逆方向に計算を進める
- Backwardの過程で各パラメータについてのエラーを計算する

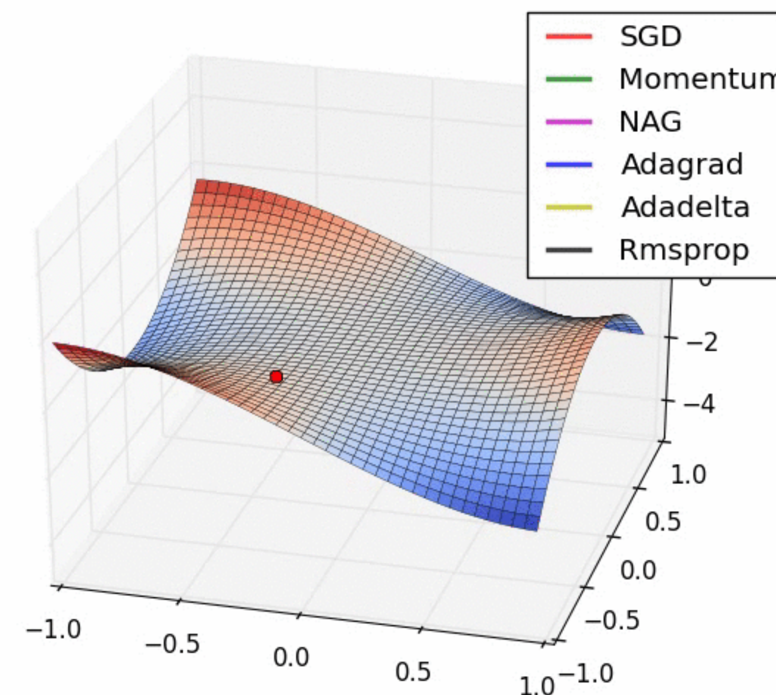
Preferred Infrastructure

パラメータの更新

- Backwardにより得られた各パラメータについての勾配δθを用いてパラメータθを更新
  - 最も単純なのは（確率的）勾配降下法
  - $\theta \leftarrow \theta - \eta\delta\theta$　　　(η：学習率)

    最も勾配が急な方向
- 更新式には様々なバリエーションがある
  - 例：過去の勾配情報も利用する、勾配（1回微分）だけでなく曲率（2回微分）も利用する
- SGD / Momentum / AdaGrad / ADADELTA / RMSprop / Adam etc…





http://imgur.com/a/Hqolpより

Preferred Infrastructure

深層学習フレームワークChainer

Preferred Infrastructure.

Chainer概要
http://chainer.org

- 製作者：得居誠也、開発：PFN、PFI
- 公開：2015年6月9日
- 隔週水曜日リリース
  - 最新バージョン：1.5.1（2015年12月10日）
- ライセンス：MIT (Expat)

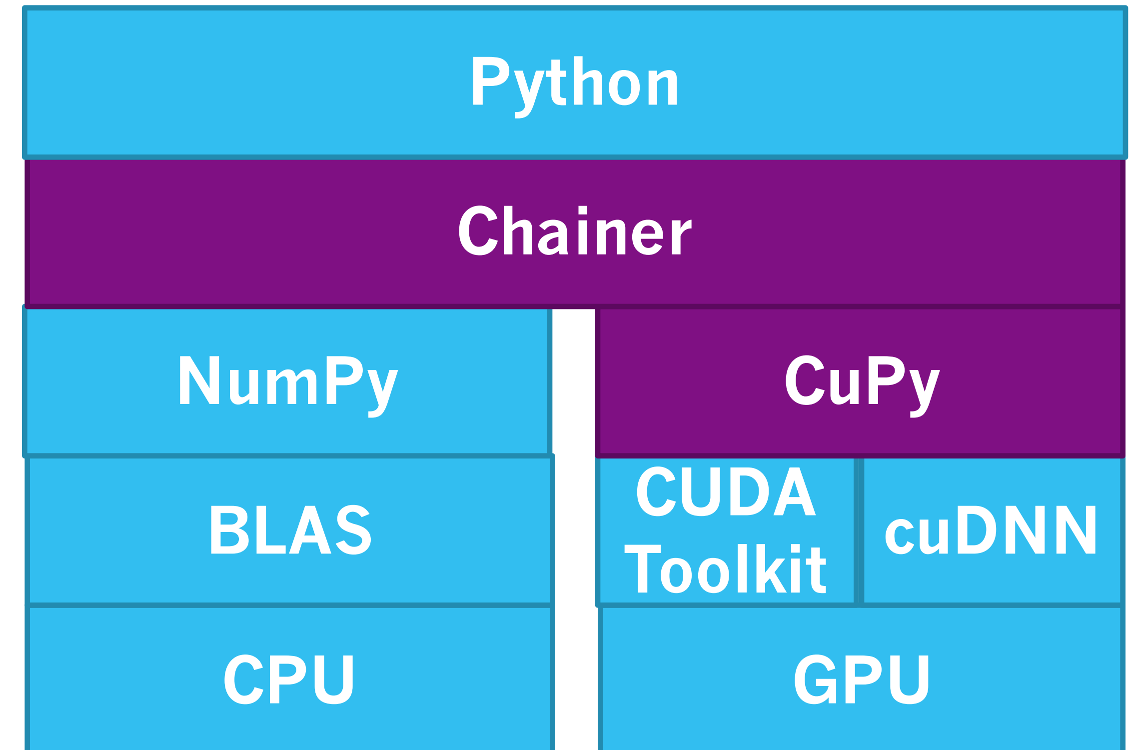
特長：様々なNNを直感的に記述可能

- NNの構築をPythonのプログラムとして記述
  - フレームワーク特有のDSLを習得しなくてよい
  - ループや分岐などを伴う複雑なNNの構築も用意
- CuPyによるCPU/GPU agnosticなコード記述
- 動的なNN構築（Define-by-Run）
  - Pythonのスタックトレースを利用したデバッグが可能
  - NNのバグがどの行で発生したかを追跡可能

Preferred Infrastructure.

Chainer技術スタック：Chainer本体 + CuPy

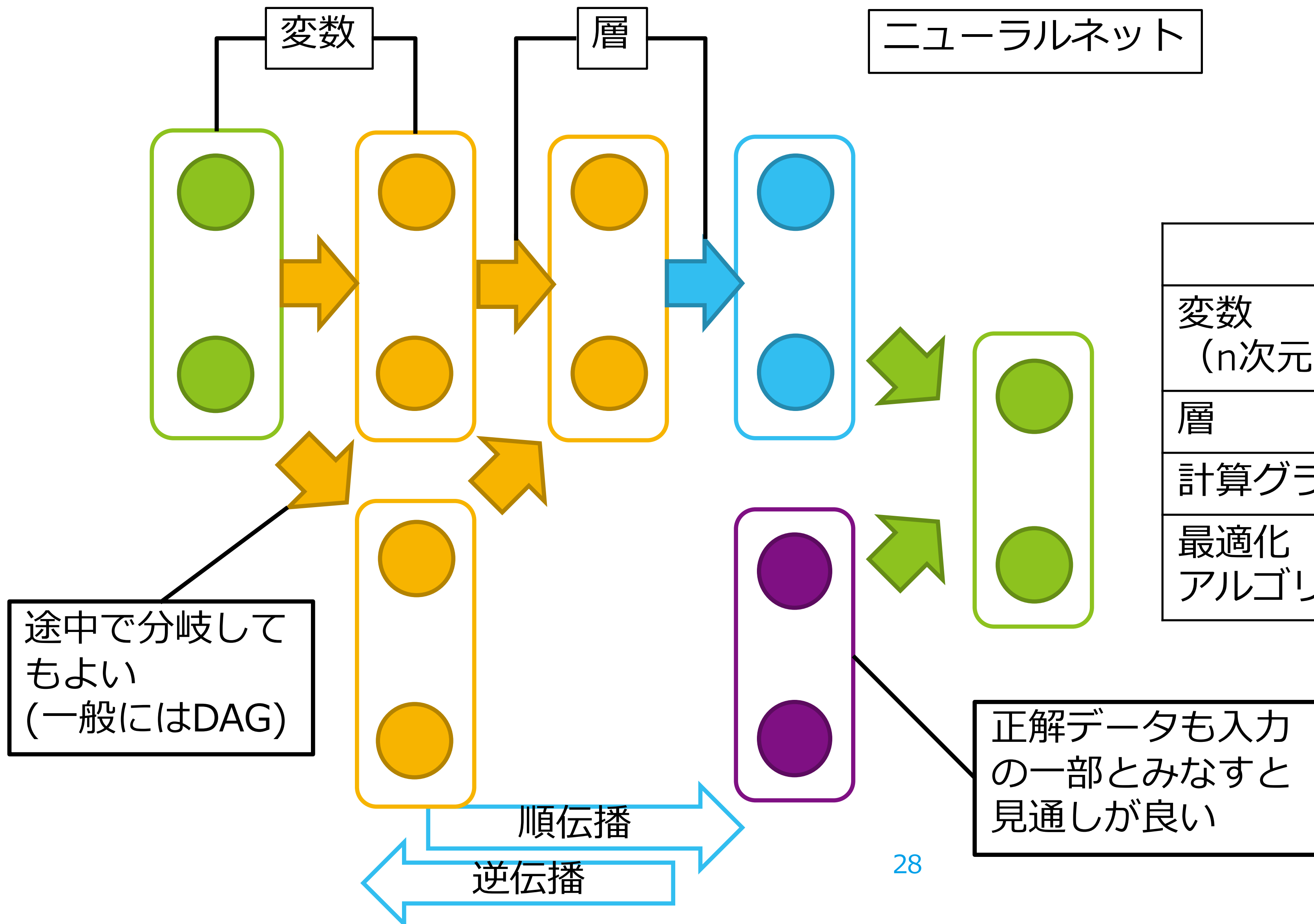
- **Chainer**：ディープラーニングフレームワーク
  - 計算グラフ構築・最適化アルゴリズムをPythonプログラムとして記述
- **CuPy**：GPU版NumPy*
  - NumPyの配列操作のサブセットと互換



* NumPy：Pythonの数値計算ライブラリ。
多次元配列の操作や数学関数が充実しており、
多くのPythonデータ解析ツールがNumPyをベースに制作されている

Preferred Infrastructure

DeepLearningフレームワークの構成要素



	Caffe	Chainer
変数（n次元配列）	Blob	Variable
層	Layer	Link
計算グラフ	Net	Chain
最適化アルゴリズム	Solver	Optimizer

Preferred Infrastructure.

入出力はテンソル（Tensor）として扱う

テンソル = 多次元配列（N次元配列）

ベクトル、行列の一般化

- 0次元テンソル＝スカラー（ただの数）
- 1次元テンソル＝ベクトル
- 2次元テンソル＝行列

プログラミング言語での記述

- `a[i][j][k] = 10`（3次元テンソル）

画像解析では4次元テンソルを扱うことが多い

- ミニバッチ x チャンネル x 縦 x 横

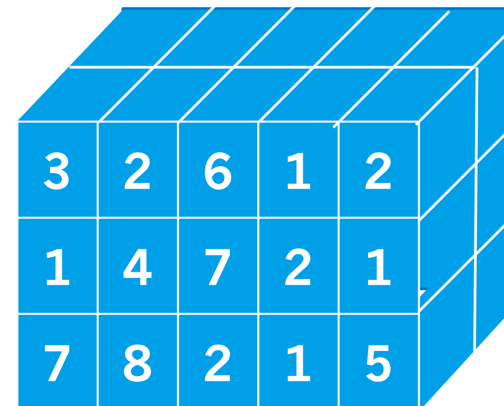
1次元テンソル

3	2	6	1	2

2次元テンソル

3	2	6	1	2
1	4	7	2	1
7	8	2	1	5

3次元テンソル



以下では、次元、サイズ、Shapeを以下の意味で使う
例：一番下のテンソル
●次元 = 3、サイズ = 3x2x2 = 12 、Shape = (3, 2, 2)

Preferred Infrastructure.

ミニバッチ：入力データをまとめたテンソル

実装では複数の入力データ（例えば128個）をまとめたテンソル（ミニバッチ）をNNに入力し、ミニバッチ内の各訓練データに対して同一の操作を行う

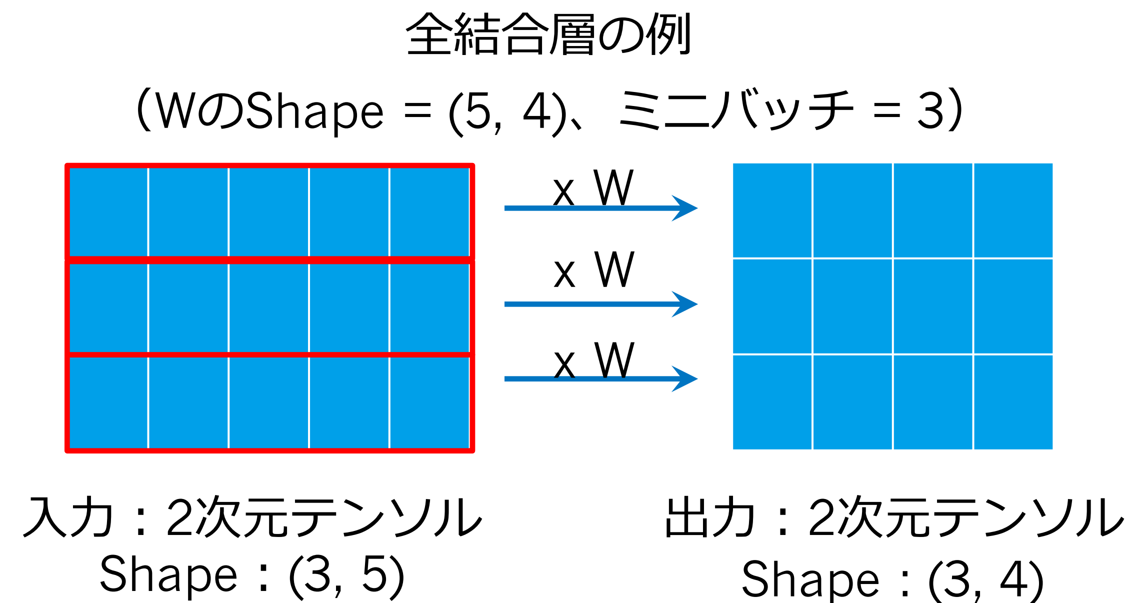
ミニバッチ化した入力は元々の入力より1次元大きい

ミニバッチ化の長所

- 操作をまとめられる分、高速化が狙える

ミニバッチ化の短所

- 使用メモリ量はミニバッチサイズに比例
- 通常GPUのメモリ容量ギリギリまでミニバッチサイズを大きくする

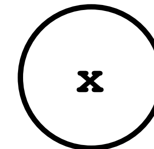


Preferred Infrastructure.

Variableオブジェクト

- 計算グラフのデータノード
- NumPyもしくはCuPyの多次元配列を保持する
- 多くのFunctionは配列の最初の軸をミニバッチとして扱う

```
x = Variable(np.zeros((10, 20), dtype=np.float32))
x.data  # Variableが格納している配列へアクセス
```

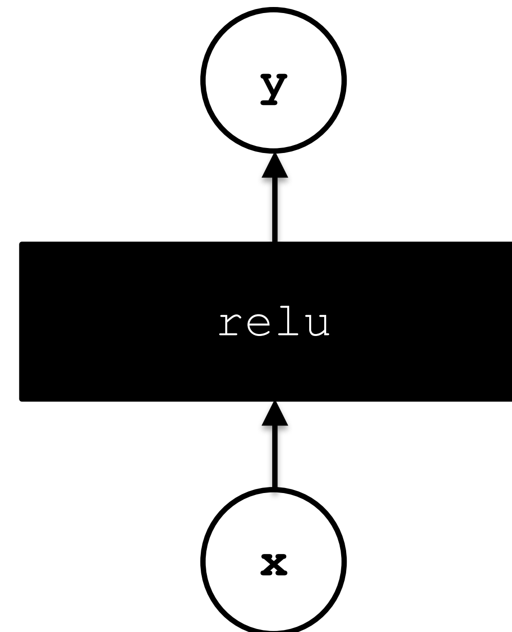


Preferred Infrastructure.

Functionオブジェクト

- 計算グラフの（パラメータを持たない）演算ノード
- `chainer.functions`（以降`F`）に色々定義されている
- FunctionはVariable（のタプル）を受け取り、Variable（のタプル）を出力する

```
x = Variable(...)
y = F.relu(x)  # yもVariable
```



Preferred Infrastructure.

Linkオブジェクト

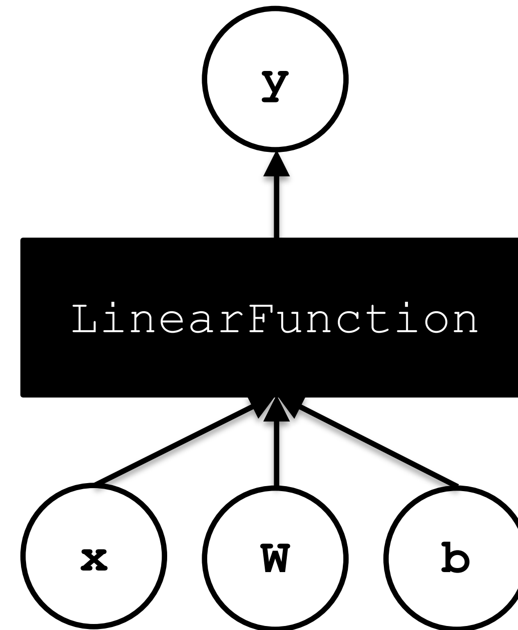
- パラメータ付き関数
- chainer.links（以降L）に定義されている
- Linkの持つパラメータは
    - Optimizer（後述）の最適化の対象
    - シリアライザでsave/loadができる
- 多くの関数では内部で対応するFunctionを利用している

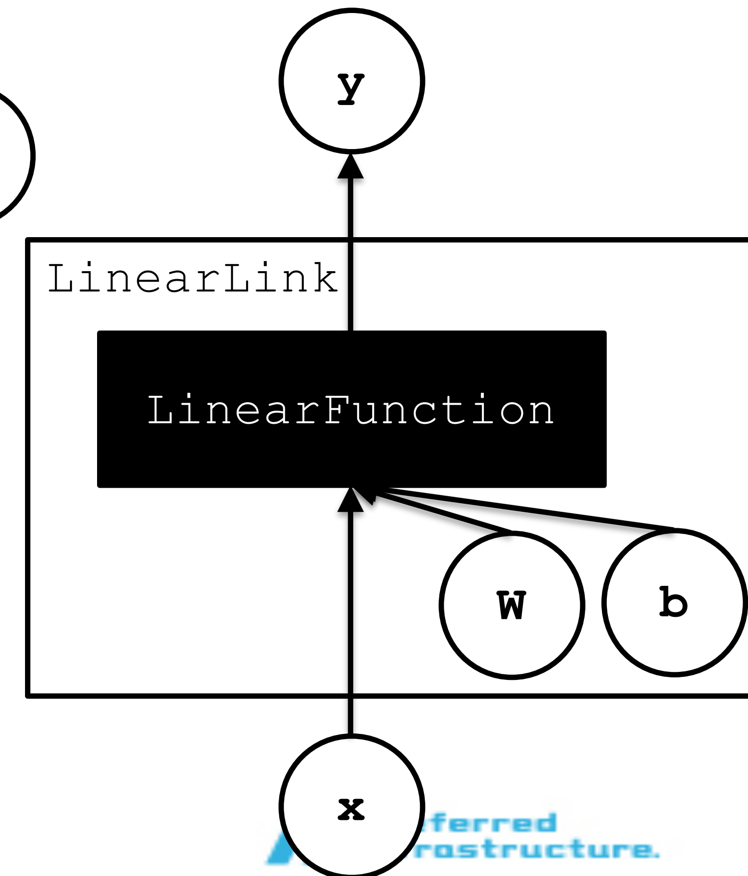
例：Linear（実際のコードとは一部異なります）→

```
class LinearFunction(chainer.Function):
    # implicitly called by self.__call__
    def forward(self, inputs):
        x, W, b = inputs
        return x.dot(W.T) + b

def linear(x, W, b):
    return LinearFunction()(x, W, b)

class Linear(chainer.Link):
    def __call__(self, x):
        return linear(x, self.W, self.b)
```

LinkとFunctionの関係

- 多くのLinkには対応するパラメータなしのFunctionが存在する
  - 例：LinearLinkとLinearFunction
- 多くのLinkは内部でFunctionを利用している
- もともとパラメータのない関数にはLinkはなくFunctionのみが存在する
  - 例：ReLUなど活性化関数の一部、プーリング関数





ChainでLinkをまとめる

- 一般にLink（パラメータ付き関数）は複数あるので、Chainでまとめて管理する
- Chain自身がLinkを継承しているので、Linkを階層的に扱える
- Chainを継承したモジュールは再利用をしやすくなる

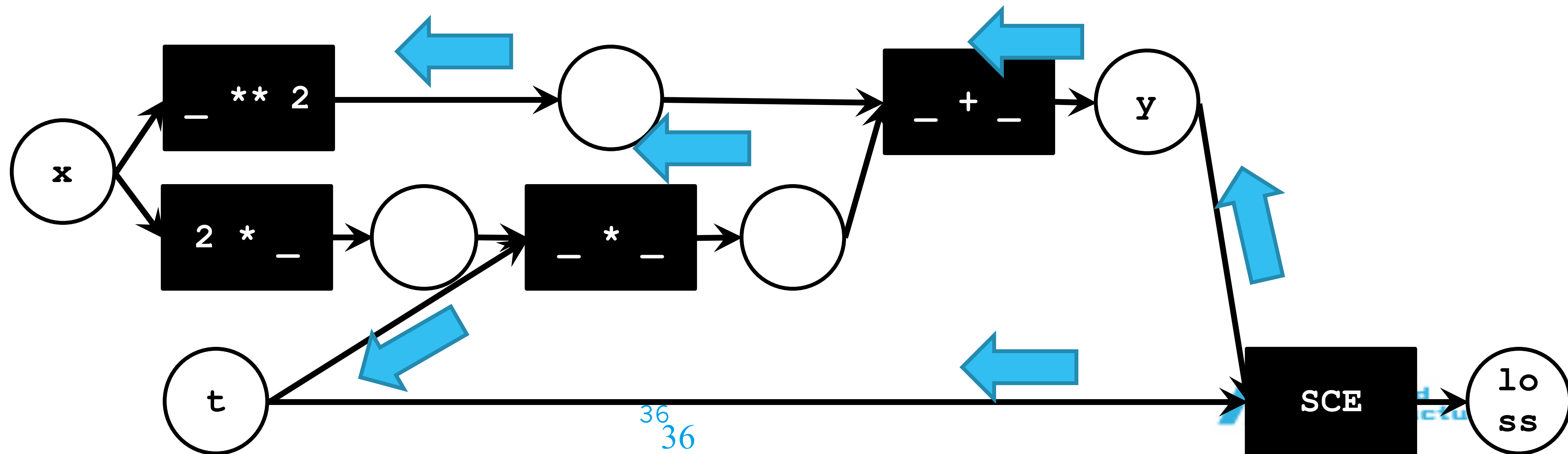
```
model = Chain(embed=L.EmbedID(10000, 100),
              layer1=L.Linear(100, 100),
              layer2=L.Linear(100, 10000))
x = Variable(...)
h = F.relu(model.layer1(model.embed(x)))
y = model.layer2(h)
```


Preferred Infrastructure.

損失関数・勾配計算

- 損失関数もFunctionの一種
- 損失関数の出力のVariableに対して、`Variable.backward()`を呼ぶと勾配が計算できる

```
loss = F.softmax_cross_entropy(y, t)

loss.backward()
```



36

Optimizerオブジェクト

- Backwardにより計算できた勾配を用いて、パラメータを最適化する
- `chainer.optimizers`に定義されている
  - 実装されている最適化法：SGD, MomentumSGD, AdaGrad, RMSprop, RMSpropGraves, AdaDelta, Adam

```
optimizer = optimizers.SGD()
optimizer.setup(model)  #最適化対象のLink/Chainをsetupで渡す
optimizer.add_hook(optimizer.WeightDecay())  # 正則化はhook関数として登録
```


Preferred Infrastructure

Optimizerによる最適化

```
model.zerograds()  # 勾配をゼロ初期化
loss = ...  # 順伝播で損失を計算
loss.backward()  # 逆伝播で勾配を計算
optimizer.update()  # パラメータを更新
```


Preferred Infrastructure

深層学習の学習まとめ

1. 目的関数を設計 = 計算グラフを構築 + 訓練データを順伝播
2. 勾配を計算：誤差逆伝播を用いて自動的に計算
3. 目的関数最小化のために、パラメータを反復更新：勾配法を用いて自動的に計算

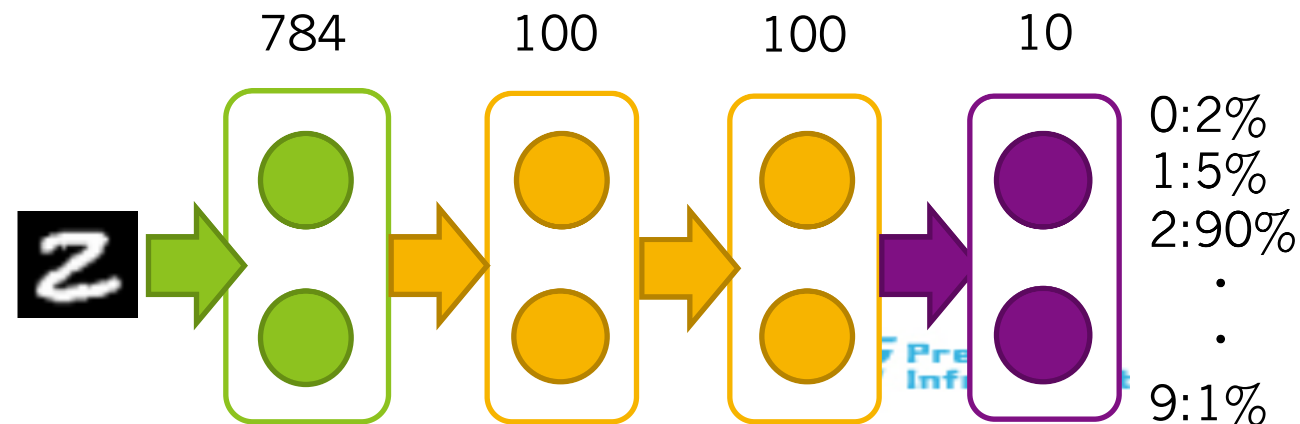
- 2, 3は深層学習フレームワークが自動的に行う
- ユーザーは1の開発に集中すれば良い

Preferred Infrastructure

例：MNISTによる多層パーセプトロンの訓練

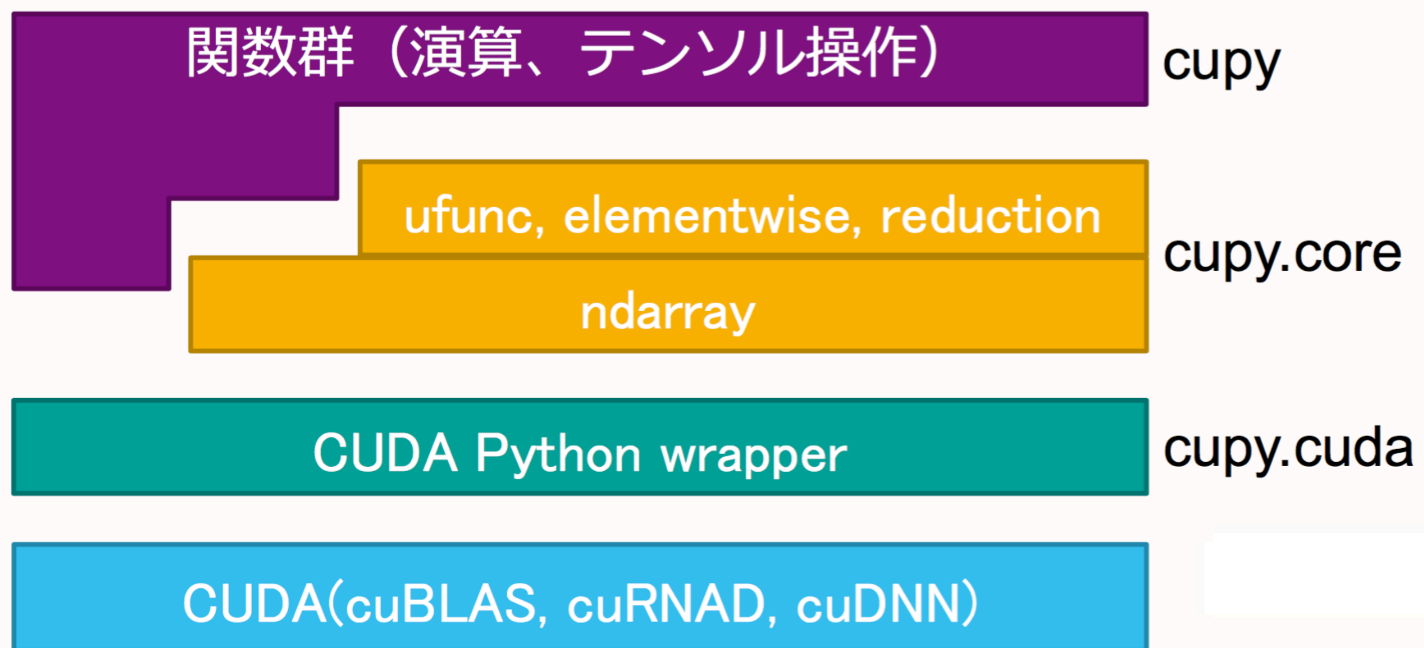
```
class MnistMLP(chainer.Chain):
 def __init__(self, n_in, n_units, n_out):
  super(MnistMLP, self).__init__(
    l1=L.Linear(n_in, n_units),
    l2=L.Linear(n_units, n_units),
    l3=L.Linear(n_units, n_out),)
 def __call__(self, x):
  h1 = F.relu(self.l1(x))
  h2 = F.relu(self.l2(h1))
  return self.l3(h2)


model = MnistMLP(784, 100, 10)
opt = optimizers.SGD()
opt.setup(model)
```

```
for epoch in xrange(n_epoch):
  for i in xrange(0, N, batchsize):
    x = Variable(to_gpu(...))
    t = Variable(to_gpu(...))
    opt.zero_grads()
    loss = model(x, t)
    loss.backward()
    opt.update()
```



CuPy：GPU版NumPy (CUDA + NumPy)

- CUDA上で計算を行うNumPyサブセットのライブラリ
  - Chainer1.5.0ではNumPy互換の関数を約170個実装済
  - 配列操作（スライス・転置・reshape等）も自由
- 簡単にGPUを扱えることを追求
- CPUコードとGPUコードの統一的な記述をサポート



CuPyの使い方

- `numpy.***` を `cupy.***` に変更すればだいたい動く
- `chainer.cuda.get_array_module`は入力に応じて`numpy`, `cupy`のいずれかを返し、CPU/GPUで共通のコードを記述できる

- 例：NumPy, CuPy両方の多次元配列（`ndarray`）を受け付けるSoftmaxの実装

```
def softmax(x)
```

```
    y = x - x.max(axis=1, keepdims=True)
```

```
    return y / y.sum(axis=1, keepdims=True)
```

CuPyでの独自Elementwiseカーネル実装

- Elementwiseな操作を行う独自カーネルの実装をサポートする仕組みを用意
- 例：2つの配列のユークリッド距離の2乗を計算する関数

```
squared_diff = cupy.ElementwiseKernel(
    'float32 x, float32 y',   #入力
    'float32 z',              #出力
    'z = (x - y) * (x - y)', #計算
    'squared_diff')          #名前
```

CuPyは内部でCUDAコードのテンプレートを持つ。引数でテンプレートを埋めて動的にコード生成、コンパイルを行う

```
diff = squared_diff(cupy.arange(10), 10))
```

10がbroadcastされて、[1, 2, 3, ..., 10]と[10, 10, ..., 10]の2乗誤差を計算

Preferred Infrastructure

CuPyでの独自Reductionカーネル実装

- Reduction操作を行う独自カーネルの実装をサポートする仕組みを用意
- 例：配列のユークリッドノルム（L2ノルム）を計算する関数

```
l2norm_kernel = cupy.ReductionKernel(
    'T x',          # 入力
    'T y',          # 出力
    'x * x',        # 前処理
    'a + b’,        # リデュース
    'y = sqrt(a)', # 後処理
    '0',            # 初期値
    'l2norm’)       # 名前
```

```
x = cupy.arange(10, dtype=‘f’)
dist = l2norm_kernel(
    x, axis=1, keepdims=True)
```


Preferred Infrastructure.

ベンチマーク（Caffeとの比較）

			ImageNetの有名な例			基礎的な畳み込みニューラルネットワーク				
			AlexNet	Overfeat	VGG	conv1	conv2	conv3	conv4	conv5
forward	Chainer	第1バッチ	59.95	76.13	49.50	24.36	22.67	26.46	19.56	19.75
	Chainer	第2~11バッチ平均	20.28	39.11	11.95	11.79	5.44	14.81	2.75	3.07
	Caffe	第1~10バッチ平均	12.30	37.26	8.45	9.43	3.46	11.88	2.45	2.84
backward	Chainer	最初のバッチ*10	61.45	73.40	55.62	27.19	20.90	26.86	20.07	19.75
	Chainer	第2-11バッチ平均	26.04	46.81	20.40	17.53	8.69	17.04	2.63	3.05
	Caffe	第1-10バッチ平均	20.52	47.70	19.17	20.36	10.60	17.34	2.97	3.13

＊ Caffe：最も使われているディープラーニングフレームワークの一つ、C++で記述され画像認識に強い

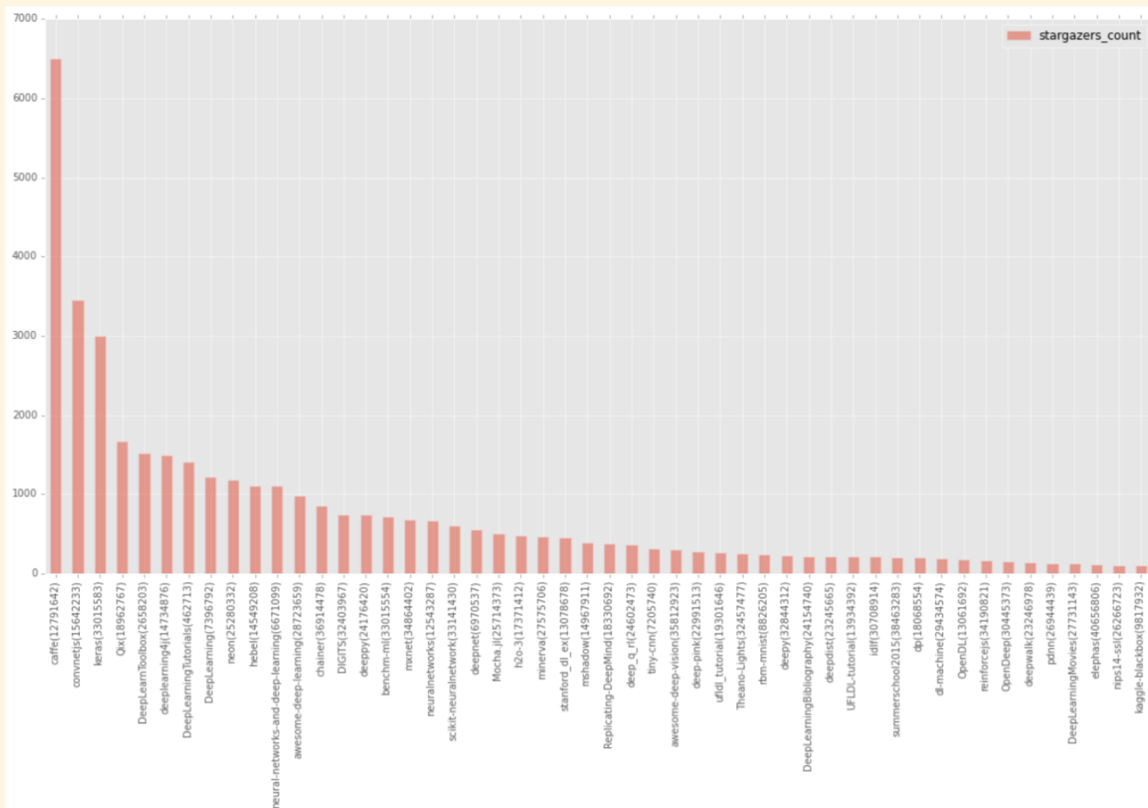
Preferred Infrastructure.

深層学習フレームワークの類別

Preferred Infrastructure.

深層学習フレームワークの乱立

DLライブラリ@GitHub



「深層学習ライブラリの現在」Yuta Kashino
http://www.slideshare.net/yutakashino/deep-learning-libs-twm

- 2014年ごろから深層学習フレームワークが乱立しはじめた
- GitHubでWatcherが2人以上(2015年10月20日現在)
  - “deep learning” : 393
  - “theano” : 239
  - “caffe” : 281

Caffe　TensorFlow™

DL4J

PyLearn2　dmlc mxnet　Chainer

計算グラフ構築のパラダイム：宣言的 vs. 命令的 [得居15]

宣言的：計算グラフをデータ記述言語で記述

- 長所
  - プログラマでないユーザにも受け入れやすい
- 短所
  - フレームワークは設定ファイルのパーザーを実装する必要がある
  - 複雑なNNの記述は困難

命令的：計算グラフを汎用プログラミング言語で記述

- 長所
  - 制御構文を用いて複雑なNNを記述可能
- 短所
  - 非プログラマには初期学習コストが高い

GoogLeNetのprototxt
2000行以上ある→

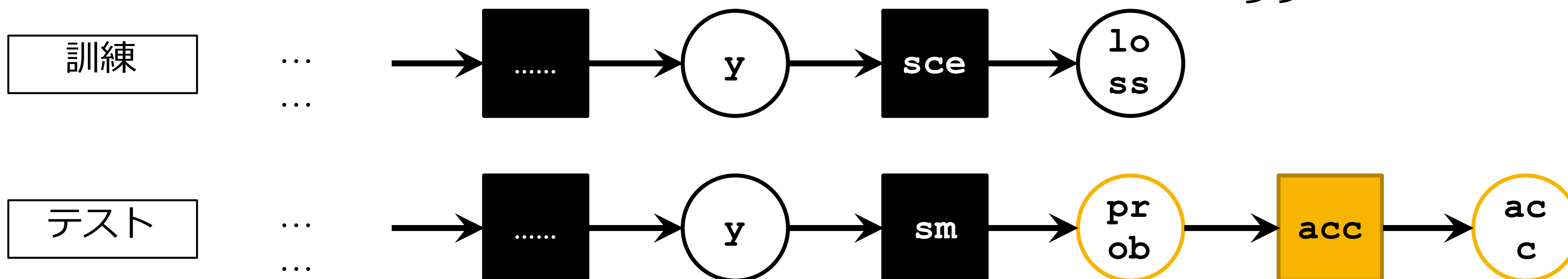
2,055 examples/googlenet/googlenet_train_test.prototxt

```
@@ -0,0 +1,2055 @@
1  +name: "GoogLeNet"
2  +layers {
3  +  name: "data"
4  +  type: DATA
5  +  top: "data"
6  +  top: "label"
7  +  data_param {
8  +    source: "examples/imagenet/ilsvrc12_train_lmdb"
9  +    backend: LMDB
10 +    batch_size: 128
11 +  }
12 +  transform_param {
13 +    crop_size: 228
14 +    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
```

命令的なフレームワークでは柔軟な計算グラフ構築が可能

```
def forward(x, t, train=True):
  h = F.relu(model.l1(x))
  y = model.l2(h)
  if train:
    loss = F.softmax_cross_entropy(y, t)
    return loss
  else:
    prob = F.softmax(y)
    acc = F.accuracy(prob, t)
    return acc
```

- ネットワーク構築時に、通常のPythonの制御構文を利用できる (if / for / while etc…)
- 応用
  - 訓練・テストで層を取り替える
  - For文を用いてRNNを構築
  - 訓練データごとに異なる計算グラフ

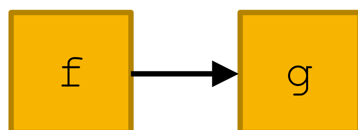


計算グラフ構築のパラダイム：Define-*and*-Run vs. Define-*by*-Run

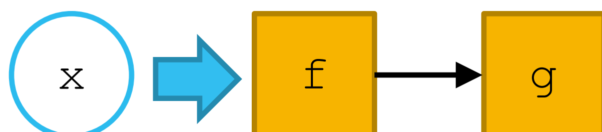
Define-*and*-Run

```
layers {
  name: "ip"
  type: INNER_PRODUCT
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
```

計算グラフ構築



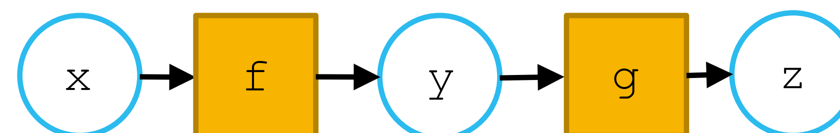
データフィード



Define-*by*-Run

```
x = chainer.Variable(...)
y = f(x)
z = g(x)
```

データフィード
= 計算グラフ構築



誤解を恐れず言えば

- Define-and-Run = NNのコンパイラ
- Define-by-Run = NNのインタープリタ

Preferred Infrastructure

Define-and-RunとDefine-by-Runの比較

Define-and-Run

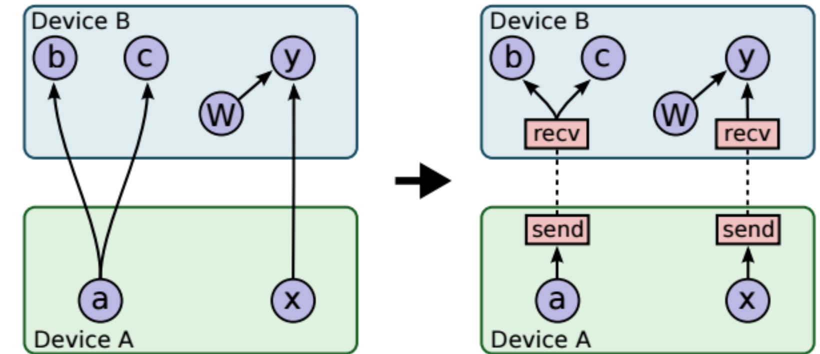
- 長所
  - 計算グラフと利用する変数が全て得られているので、計算グラフの最適化が容易
- 短所
  - 全訓練データで計算グラフが共通でなければならない
  - データと計算グラフの不整合によるバグを追求しづらい

Define-by-Run

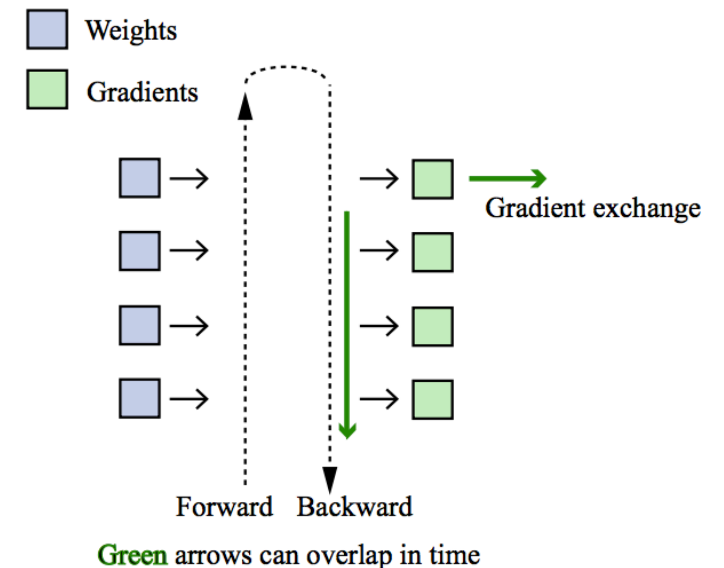
- 長所
  - データと計算グラフの不整合によるエラーを特定しやすい
  - データに依存した計算グラフを構築可能
- 短所
  - 計算グラフの最適化が困難

Define-and-Runのパラダイムでは、計算グラフを最適化しやすい

- Theanoではコンパイル時に合成可能なoperatorの合成を行っている
- Caffeでは勾配が不要な箇所でのBackward操作を省略している
- Caffeでは多くの活性化関数の計算はin-placeで行い、メモリ消費量を節約している
- Tensorflowでは計算グラフを複数ノードで操作するとき、計算グラフ内のそれぞれのoperatorの計算コストを推定し、ノードへ適切に配置する
- Purine2ではNNの上部のパラメータ交換とNN下部でのbackwardを同時に行う事でGPU上での計算を転送で隠蔽している



↑[Abadi+15]　↓[Lin+14]



Chainerの最適化面での課題

- Define-by-RunパラダイムであるChainer上でいかに計算グラフの最適化を行うか？
  - CuPyの生成するカーネルの最適化・カーネル合成
  - 計算グラフを構成するFunctionの合成
- 最適化とデバッグの容易さのトレードオフ
- Define-and-RunとDefine-by-Runのいいとこ取りの“NNのJITコンパイラ”のようなものがあれば良い？

Preferred Infrastructure.

まとめ

- 2012年のコンペティション以後、深層学習は画像・音声・自然言語など様々な分野で利用されています
- ChainerはPythonベースの深層学習フレームワークで、Define-by-Runパラダイムを採用することで柔軟な計算グラフ構築をプログラムとして記述できます

- 公式HP：http://chainer.org
- レポジトリ： https://github.com/pfnet/chainer
- Twitter：@ChainerOfficial
- Google Group：Chainer Uesr Group
- Contribution Guide：http://docs.chainer.org/en/stable/contribution.html

We are hiring!

参考文献

[Krizhevsky+12] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[Vinyal+14] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2014). Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*.

[Le, Ng, Jeffrey+12] Le, Q. V. (2013, May). Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (pp. 8595-8598). IEEE.

[Taigman+14] Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014, June). Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (pp. 1701-1708). IEEE.

[Hannun+14] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... & Ng, A. Y. (2014). DeepSpeech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

[得居15]ニューラルネットワークの実装 深層学習フレームワークの構成と Chainer の設計思想、人工知能学会

[Abadi+15]Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*.

[Lin+14]Lin, M., Li, S., Luo, X., & Yan, S. (2014). Purine: A bi-graph based deep learning framework. *arXiv preprint arXiv:1412.6249*.

Preferred Infrastructure

Copyright © 2014-

Preferred Networks All Right Reserved.

Preferred Networks

深層学習の最適化の課題

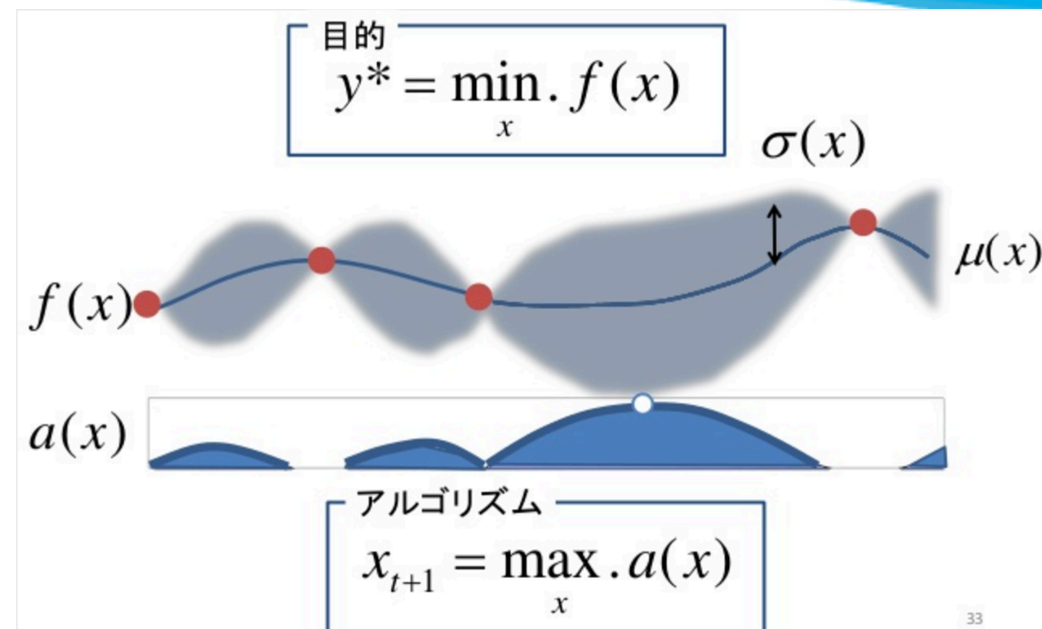
Preferred Infrastructure.

深層学習は設計の自由度が高い = チューニングが難しい

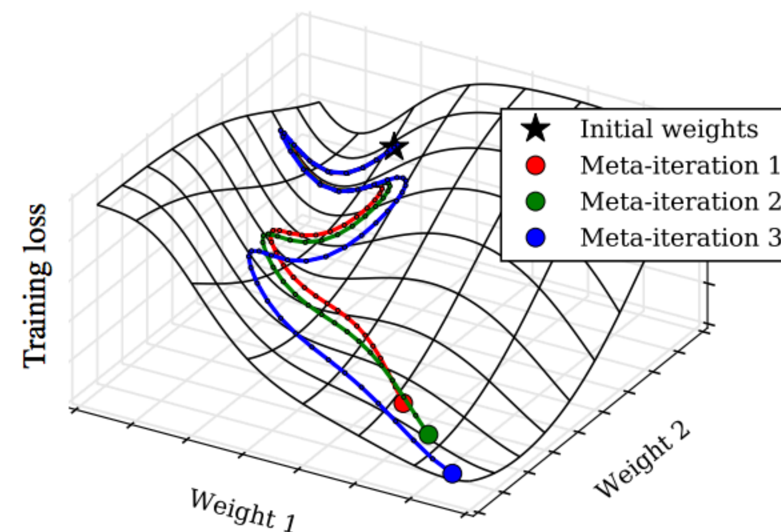
- 設計時の選択肢：NNの設計は回路設計に近い
  - ネットワーク（トポロジー/Layer数/Node数/活性化関数/損失関数）
  - 学習方法（学習アルゴリズム/Iteration数/学習率関連）
- チューニングパラメータが精度に大きく影響を与える
  - ReLUで少し学習率を変えただけで損失がInfになる
  - アルゴリズムの改善よりもパラメータ探索が重要になることも
  - パラメータチューニングのコツをまとめた論文もある[Bengio12][Hinton12]
- チューニング対象が特徴抽出からハイパーパラメータに変わっただけ？

ハイパーパラメータチューニングの（全 / 半）自動化

- Spearmint : ガウス課程を用いたベイズ最適化(BO)による機械学習アルゴリズムのハイパーパラメータチューニング [Snoek+12]
  - CNN + BOによる病変検出 [佐藤+14, Unpublished?]
- NN + ベイズロジスティック回帰によるBO[Snoek+15]
- 勾配法によるNNのハイパーパラメータ最適化 [Maclaurin+15]
- maf : 機械学習の実験ビルドツール
  - 複数の（ハイパー）パラメータでの実験の一括実行・中間生成物・実験結果の管理



↑[佐藤+14]　↓[Maclaurin+15]



mafを用いた実験例：liblinear

```
def build(bld):
    NUM_FOLD = 10
    bld(source='data/svmguide3',
        target='data/train data/test',
        parameters=[{'fold': i} for i in range(NUM_FOLD)],
        rule=maflib.rules.segment_by_line(NUM_FOLD, 'fold'))
    bld(source='data/train',
        target='data/model log/train',
        parameters=maflib.util.product({
            'type': [1, 2, 3, 4, 5, 6, 7],
            'cost': [0.01, 0.1, 1, 10, 100]}),
        rule='liblinear-train -s ${type} -c ${cost} ${SRC} ${TGT[0].abspath()} > ${TGT[1].abspath()}')
    bld(source='data/test data/model',
        target='result/predict log/test',
        rule='liblinear-predict ${SRC} ${TGT[0].abspath()} > ${TGT[1].abspath()}')
    bld(source='log/test',
        target='result/accuracy',
        rule=maflib.rules.convert_libsvm_accuracy)
    bld(source='result/accuracy',
        target='result/average_accuracy',
        aggregate_by='fold',
        rule=maflib.rules.average)
    bld(source='result/average_accuracy',
        target='result/plot.png',
        for_each=[],
        rule=plot)
```

パラメータの作成

パラメータの作成

パラメータの集約

パラメータの集約

交差検定用にデータ分割

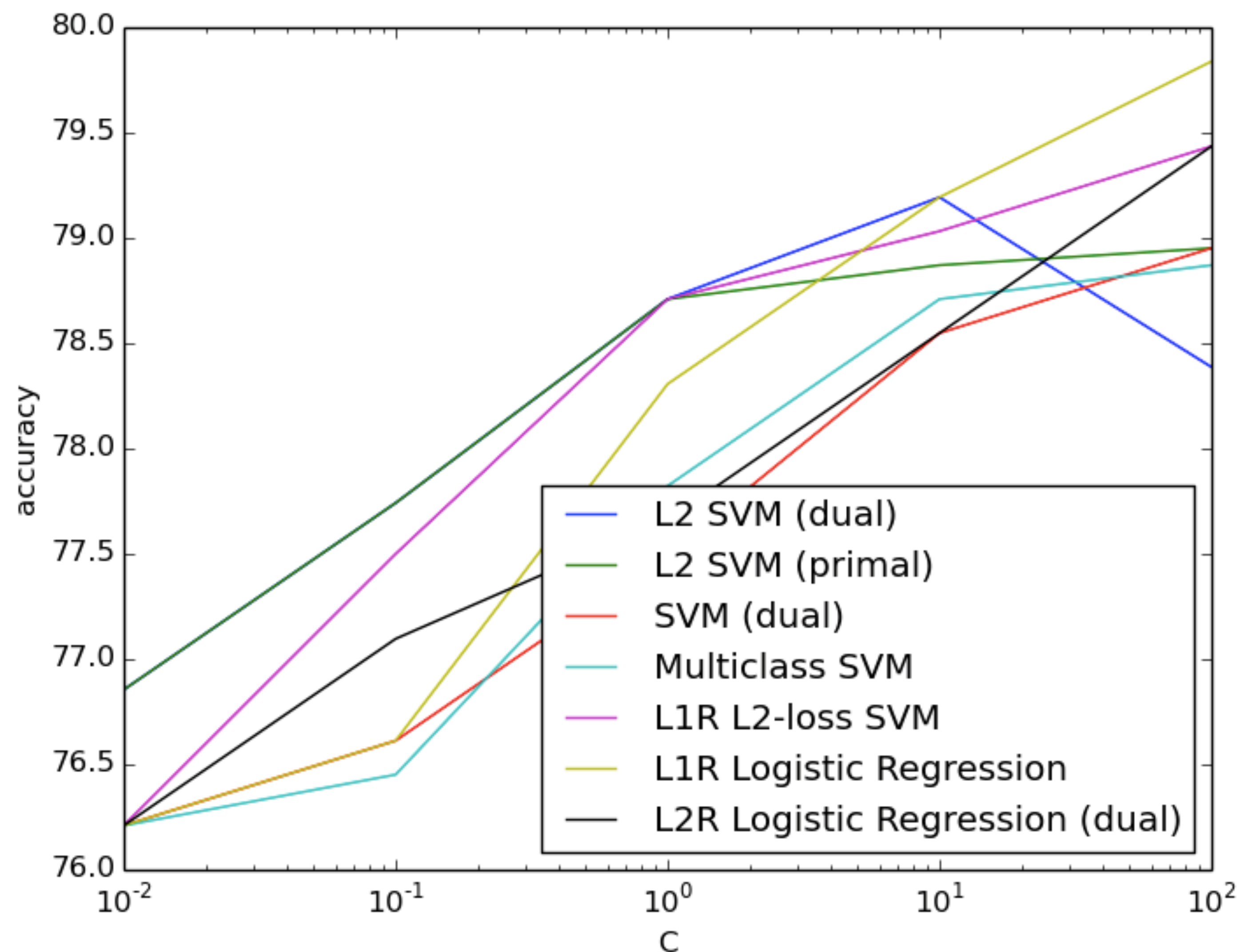
訓練

テスト

集計

プロット

mafデモ：実行結果例



Preferred Infrastructure.

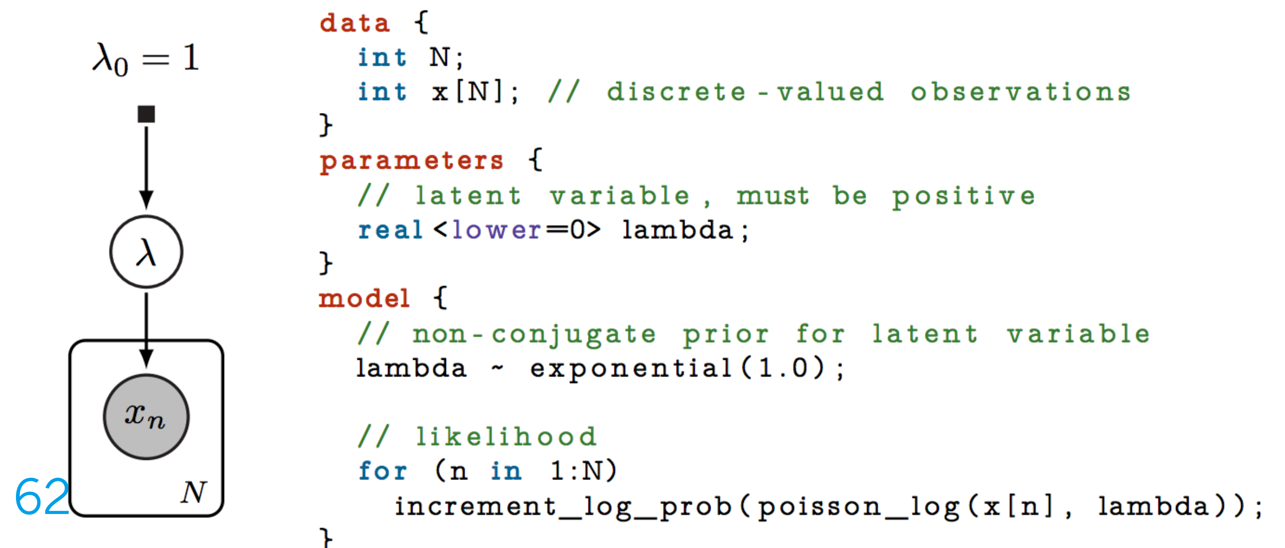
変分ベイズ近似の自動化[Kucukelbir+14]

- 変分ベイズ近似
  - 生成モデル $p(x|z;\theta)$ と観測が与えられている(x:変数, z:変数, θ：生成モデルのパラメータ)
  - 観測データ $\mathbf{X} = \{x1, \ldots, xn\}$ に対し、事後確率 $p(z|\mathbf{X})$ を計算したいが通常計算できない
  - $p(z|\mathbf{X})$ を計算しやすい確率分布族 $q(z;\varphi)$ で近似する（φ：推論モデルのパラメータ)
  - θ, φを同時に最適化する
- 最近だと $p(x|z;\theta)$, $q(z;\varphi)$ を両方NNでモデル化してしまう事が多い

- 生成モデルをStan（確率的プログラミング言語）で記述すると、推論モデルの構築と最適化を自動的に行う

$$p(\mathbf{X}) \geqq \mathbf{E}_{q(z|\mathbf{X})}[p(\mathbf{X}|z)] + KL(q(z)||p(z|\mathbf{X}))$$

等号成立条件：$p(z|\mathbf{X}) = q(z) \quad \forall z$



まとめ

- 機械学習・深層学習は最適化のための道具として利用されることが多いですが、計算グラフの効率的な実行・ハイパーパラメータチューニング・変分ベイズ近似の推論モデルの設計など、機械学習・深層学習自体にもの最適化の余地があります
- ハイパーパラメータ探索を全自動化する方法として、ベイズ最適化の手法が研究されています
- mafは様々なハイパーパラメータでの実験を一括して実行・管理するソフトウェアです

参考文献

[Bengio12]Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade* (pp. 437-478). Springer Berlin Heidelberg.

[Hinton12]Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade* (pp. 599-619). Springer Berlin Heidelberg.

[Snoek+12]Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951-2959).

[佐藤14]佐藤一誠、Deep Learningによる医用画像読影支援、http://www.slideshare.net/issei_sato/deep-learning-41310617

[Maclaurin+15]Dougal Maclaurin, David Duvenaud, Ryan Adams Gradient-based Hyperparameter Optimization through Reversible Learning, Proceedings of The 32nd International Conference on Machine Learning, pp. 2113–2122, 2015

[Snoek+15]Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., ... & Adams, R. P. (2015). Scalable Bayesian Optimization Using Deep Neural Networks. *arXiv preprint arXiv:1502.05700*.

[Kucukelbir+14] Kucukelbir, A., Ranganath, R., Gelman, A., & Blei, D. (2015). Automatic variational inference in Stan. In *Advances in Neural Information Processing Systems* (pp. 568-576).